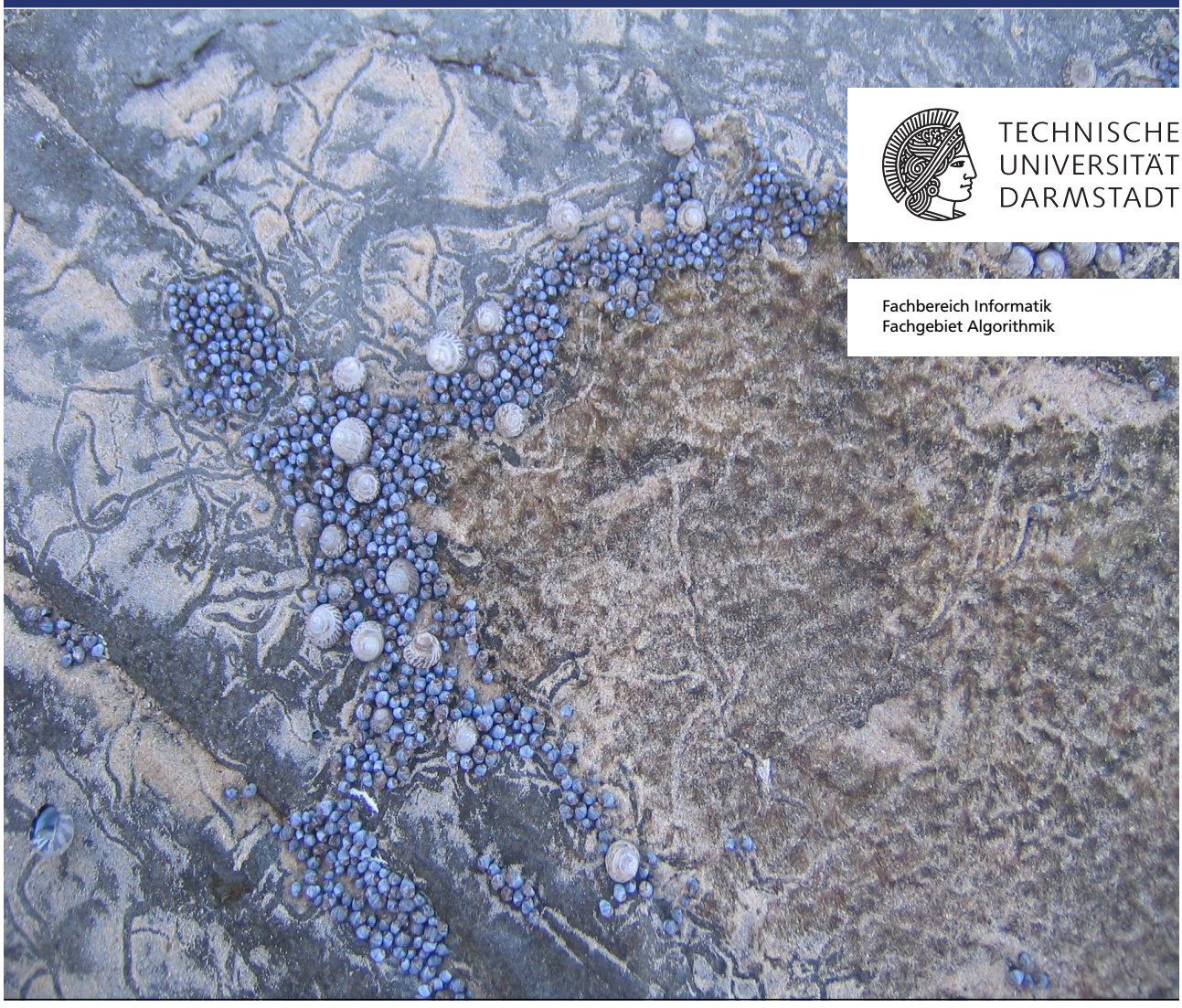
Solution Techniques for specific Bin Packing Problems with Applications to Assembly Line Optimization

Zur Erlangung des Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.) vom Fach–bereich Informatik genehmigte Dissertation von Dipl.-Math. Wolfgang Stille aus Göppingen
Juni 2008 — Darmstadt — D 17



TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Fachgebiet Algorithmik

Solution Techniques
for specific
Bin Packing Problems
with Applications to
Assembly Line Optimization

vom Fachbereich Informatik genehmigte Dissertation von Dipl.-Math. Wolfgang Stille aus Göppingen

1. Gutachten: Prof. Dr. Karsten Weihe
2. Gutachten: Prof. Dr. Matthias Müller-Hannemann

Tag der Einreichung: 19. Mai 2008
Tag der Prüfung: 30. Juni 2008

Darmstadt — D 17

Abstract

The present thesis is about efficient solution techniques for specific BIN PACKING problems and their derivatives. BIN PACKING problems arise in numerous applications from industry and economics. They occur for example in the distribution of resources, in operation scheduling of complex processes, in project planning, and logistics, only to mention a few. As one concrete application, we present the optimization of assembly lines for printed circuit board manufacturing in detail. This work arose from a long term cooperation with Philips Assembléon in Eindhoven.

The BIN PACKING decision problem asks the question whether – given a set of objects of distinct sizes, and a set of bins with specific capacity – there is a distribution of items to bins such that no item is left unpacked nor the capacity of any bin is exceeded. The corresponding optimization problem searches for the minimum number of bins to pack all objects.

In general, BIN PACKING is $\mathcal{NP}$-hard. As long as $\mathcal{P} \neq \mathcal{NP}$, this amounts to the fact that there is no algorithm that is able to decide whether a feasible distribution for a given instance exists in time that depends polynomially on the size of the input. However, there are several special cases that may be solved efficiently, either approximately or even optimally. In practice, there are for example problems that comprise only few distinct item sizes, but items of each size are occurring in high multiplicities. The thesis gives theoretical insights for exactly this case. We develop an efficient combinatorial algorithm which solves the problem in polynomial time to a solution that is using at most one more bin than an optimal solution.

Moreover, we introduce various rounding techniques for rounding arbitrary input data. The presented techniques are general enough to be applied to various optimization problems, either for the computation of approximate solutions, or for the purpose to efficiently generate upper and lower bounds in order to narrow the solution space. This can be achieved for example by applying an efficiently solvable special case as described above. In order to control the impact of the relaxation, we put special emphasis on the bounding of the rounding error: for any of the presented rounding algorithms, we prove an error estimation. Furthermore, we present a comprehensive qualitative analysis on typical data profiles as well data from real world applications.

As an application, we use rounding as a relaxation technique in order to jointly evaluate a global constraint with an arbitrary number of elementary constraints. This problem arises from *Constraint Programming* which has steadily increased in popularity during the last years for modeling and solving optimization problems. We develop a framework to efficiently evaluate *Bin Packing Constraints* jointly with a class of fairly general constraints which we call *Concave Constraints*. These imply a wide variety of elementary constraints as logic operations such as implications and disjunctions, as well as all linear constraints. Moreover, arbitrary concave functions such as quadratic constraints, and constraints modeling a process of growth or decay can be modeled within this framework. We give various examples for the modeling of fundamental optimization problems within this framework. Finally, we develop algorithms that detect infeasibility of a given constraint system.

The last chapter is devoted to a concrete application: the optimization of assembly lines for printed circuit board manufacturing. Due to high–mix low–volume batches the production process must be more flexible than ever, and unprofitable setup times should be avoided as far as possible. For this reason, *partly–combined processing* is introduced: an assembly line consists of multiple trolleys that

hold the component types. One or more of these trolleys might be exchanged on the fly during the batch which amounts to a partial change in the machine setup. Changing the setup several times during the batch process implies a partition of the boards in the batch into several groups. There are two problems arising from this approach: (1) how to find an ideal grouping of boards, and (2) how to assign component types to placement modules in order to exploit the high parallelism of the assembly line. In particular, it must be decided which component types to assign statically, and which ones on the exchangeable trolleys. This problem can be understood as a generalized *Bin Packing* problem with additional side constraints. We show that the problem is $\mathcal{NP}$-complete.

In the scope of an industrial cooperation project with Philips Assembléon in Eindhoven, we developed specific algorithms in order to efficiently compute groupings of boards and machine setups for the partly–combined case. We have made several computational experiments with original data sets from customers and compared our technique to another approach from the literature. The results show that our approach is vastly superior with respect to the solution quality. Moreover, it contains some additional benefits: the computed setups are feasible in any case, and infeasibility is detected at an early stage of the framework. Additionally, this results in a much shorter runtime of the optimization software.

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit Techniken zur Lösung von speziellen BIN PACKING Problemen und deren Verwandten. BIN PACKING Probleme treten in zahlreichen Anwendungen in Industrie und Wirtschaft auf, beispielsweise bei der Verteilung von Ressourcen, der Ablaufplanung komplexer Vorgänge, im Projektmanagement und in der Logistik, nur um einige zu nennen. Als konkrete Anwendung widmen wir uns im letzten Kapitel der Optimierung von Fertigungslinien für die Leiterplattenbestückung. Diese entstand aus einer langjährigen Industriekooperation mit Philips Assembléon in Eindhoven.

Das BIN PACKING Entscheidungsproblem stellt die Frage, ob zu einer gegebenen Menge von Objekten verschiedener Größe und einer Menge von Containern mit bestimmtem Fassungsvermögen eine Verteilung der Objekte auf die Container existiert, so daß weder ein Objekt unverpackt bleibt, noch die Kapazität eines Containers überschritten wird. Das korrespondierende Optimierungsproblem sucht nach der minimalen Anzahl an Containern, so daß alle Objekte verpackt werden können.

Dieses Problem ist im allgemeinen Fall $\mathcal{NP}$-vollständig. Solange $\mathcal{P} \neq \mathcal{NP}$ bedeutet dies, daß es kein Verfahren gibt, welches die Entscheidung, ob für eine gegebene Instanz eine zulässige Verteilung existiert oder nicht, innerhalb einer Zeit treffen kann, die polynomial von der Eingabegröße des Problems abhängt. Für einige Fälle, in denen die Eingabedaten spezielle Strukturen aufweisen, kann das Problem jedoch effizient, d.h. in Polynomialzeit exakt gelöst bzw. approximiert werden. In der Praxis treten beispielsweise oftmals Probleme auf, die sehr viele Objekte beinhalten, die jedoch nur sehr wenige unterschiedliche Größen aufweisen. Für genau diesen Fall werden in dieser Arbeit die notwendigen theoretischen Grundlagen erarbeitet und ein effizientes kombinatorisches Lösungsverfahren entwickelt. Das Verfahren berechnet in polynomialer Zeit Lösungen, die maximal einen Behälter mehr benötigen als die Optimallösung.

Darüberhinaus werden Verfahren zur Rundung beliebiger Eingabedaten vorgestellt. Im allgemeinen sind solche Verfahren für Optimierungsprobleme unterschiedlichster Art anwendbar: sei es um effizient obere und untere Schranken zu berechnen oder um Näherungslösungen zu generieren, zum Beispiel indem ein effizient lösbarer Spezialfall angewandt werden kann. Da eine Rundung der Eingabedaten einer Relaxierung des Originalproblems entspricht, legen wir besonderes Gewicht auf die Beschränkung des Rundungsfehlers. Wir beweisen für jedes der vorgestellten Verfahren eine Fehlerabschätzung und präsentieren umfangreiche Rechenergebnisse auf typischen Datenprofilen.

Als eine Anwendung der Rundung stellen wir ein Verfahren zur Auswertung von *Bin Packing Constraints* zusammen mit konkaven Bedingungen vor. Diese Anwendung kommt aus dem *Constraint Programming* welches sich in den letzten Jahren immer größerer Beliebtheit bei der Lösung von Optimierungs- und Entscheidungsproblemen erfreut. Konkave Bedingungen beinhalten sowohl eine Vielzahl elementarer Constraints wie logische Verknüpfungen, Implikationen und Ausschlüsse, als auch lineare Nebenbedingungen. Darüberhinaus sind beliebige konkave Funktionen verwendbar, wie sie beispielsweise in Wachstums- und Zerfallsprozessen vorkommen. Wir präsentieren umfangreiche Beispiele, wie grundlegende Optimierungsprobleme innerhalb dieses Paradigmas modelliert werden können. Desweiteren entwickeln wir Algorithmen, die die Unzulässigkeit eines gegebenen Constraintsystems feststellen.

Das letzte Kapitel beschäftigt sich mit der Optimierung von Produktionslinien zur Platinenbestückung. Die Fertigung muß den immer kürzer werdenden Entwicklungszyklen und der enormen Bandbreite an verschiedenen Artikeln nachkommen und eine flexible Produktion ermöglichen. Damit verbunden sollen unprofitable Rüstzeiten nach Möglichkeit vermieden werden. Daher wird versucht, mit einer Komponentenbelegung möglichst viele verschiedene Platinen zu produzieren, während kleinere Variationen im Setup der Maschine während des Produktionsprozesses mithilfe spezieller austauschbarer Rollwagen, welche die Komponenten beherbergen, vorgenommen werden können. Diese Verfahrensweise impliziert eine Aufteilung der Menge von Platinentypen in verschiedene Gruppen. Zwischen den einzelnen Gruppen findet eine Variation eines Teils des Setups statt, während der Großteil des Setups jedoch über den kompletten Produktionsprozeß hinweg bestehen bleibt. Die Schwierigkeit bei dieser Vorgehensweise besteht zum einen darin, eine möglichst ideale Gruppierung von Modulen zu finden, zum anderen darin, die Bauteile so auf die parallelen Bestückungsautomaten zuzuweisen, daß diese möglichst gut ausgelastet sind. Dieses Problem kann als Verallgemeinerung des BIN PACKING Problems mit zusätzlichen Nebenbedingungen verstanden werden. Wir zeigen, daß dieses Problem $\mathcal{NP}$-vollständig ist.

Im Rahmen einer Kooperation mit Philips Assembléon in Eindhoven wurden spezielle Algorithmen zur Gruppierung von Modulen und zur effizienten Berechnung von Maschinensetups für den oben genannten Fall entwickelt. Die entstandenen Verfahren wurden an realen Daten aus Kundenaufträgen getestet und mit anderen aus der Literatur bekannten Verfahren verglichen. Dabei stellte sich heraus, daß die Anwendung der entwickelten Algorithmen neben gewisser anderer Vorteile zu einem wesentlich höheren Produktionsdurchsatz führt. Darüberhinaus konnte die Laufzeit der Optimierungssoftware um ein Vielfaches verkürzt werden.

Acknowledgment

At this point, I would like to thank several persons who played an important role in the concretization of this project, directly as well as indirectly. I am very grateful to have enjoyed or – even better – still enjoy your company. If I forgot to mention someone here, don't take it too serious.

First of all, I want to thank my supervisor, Professor Karsten Weihe. I am very much obliged to you for your inspiration, your patience, and your constant availability. I would like to thank you for the numerous opportunities you provided me.

I am deeply grateful to Professor Matthias Müller–Hannemann. Thank you for the various encouraging discussions we had over the past years. I appreciated your attendance at TU Darmstadt very much.

I would like to thank my dear colleagues for the fruitful exchange of ideas, their helpful suggestions, and their friendship.

Thanks to the people at the Department of Computer Science and the Department of Mathematics at TU Darmstadt. Over the years, I have made the acquaintance of many interesting, congenial and likable people there, and it is still going on!

Dinkum thanks go to the people at the Department of Electrical Engineering and Computer Science at the University of Newcastle, Australia. I will never forget the awesome time we have spent down under.

The work in Chapter 5 arose from a long term cooperation with Philips Assembléon in Eindhoven. I would like to thank the people there, especially Harold for interesting discussions, and Philips Assembléon for providing problem instances for scientific use and some illustrations used in this thesis.

Very special thanks are to my fantastic friends for making me smile a billion times.

Finally, I am deeply grateful to my parents. The enumeration of reasons motivating my gratitude to them would have a complexity that is exponential in the days that passed since I have begun to see the light of day. Therefore, I will desist from an explicit implementation here. Certainly, cordial thanks go to my awesome brother. Special thanks also to my parents-in-law.

Last but not least, my most whole–hearted thanks are to my beloved wife and our gorgeous son. You are the sunshine of my life. I dedicate this work to you.

Darmstadt, in May 2008

Wolfgang

Contents

1 Introduction

Ultimately, you must forget about technique.
The further you progress, the fewer teachings there are.
The great path is really no path.

— Morihei Ueshiba - O´ Sensei

1.1 Motivation and Scope

Computer based optimization and decision support are omnipresent in the industrial societies of today. In a huge number of practical problems, packing problems play a fundamental role – either as the central issue or at least as a subproblem in a more comprehensive framework. Packing problems are ubiquitous. They might be self-evident as, for example, the packing of goods into containers is when organizing the dispatch in mail order businesses or optimizing the loading of air-shipped freight. In most applications, they are not exactly obvious: packing problems occur in production planning, scheduling, resource distribution, partitioning, printed circuit board design, load balancing, vehicle–routing and delivery problems, only to mention a few. Various economical and industrial problems might be formulated as packing problems, so their importance is utmost.

The mother of all packing problems is the BIN PACKING PROBLEM, which asks the following question:

Given a set of items, each of a specific size, and a set of bins,
each of a specific size as well – is there a distribution of items to bins
such that no item is left unpacked and no bin capacity is exceeded?

The corresponding optimization problem asks for the minimum number of bins of identical capacity C such that all items can be packed. It might be formulated by the following Integer Linear Program (ILP).

$$(BP) \qquad \text{minimize} \quad \sum_{j=1}^{n} y_j \tag{1.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} w_i x_{ij} \leq C \cdot y_j, \qquad \forall j \in \{1,\ldots,n\}, \tag{1.2}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad \forall i \in \{1,\ldots,m\}, \tag{1.3}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i \in \{1,\ldots,m\},\ \forall j \in \{1,\ldots,n\}, \tag{1.4}$$

$$y_j \in \{0,1\}, \qquad \forall j \in \{1,\ldots,n\}. \tag{1.5}$$

Given are m items of size w_ℓ each, $\ell \in \{1, \ldots, m\}$, and at most n bins of capacity C each. Not every bin might be utilized in a solution. Therefore, the $0/1$–variables y_j determine whether a bin j is used or not. We also make use of assignment variables x_{ij}: x_{ij}=1, if an item i is assigned to a bin j, and 0 otherwise. The objective function (1.1) is to minimize the number of bins items are assigned to. Condition (1.2) ensures for each bin that its capacity is not exceeded by the sum of item sizes of the items assigned to this bin. This is often called the *capacity constraint* or the *knapsack constraint* as it is fundamental for the KNAPSACK PROBLEM (KP) as well. Condition (1.3) ensures that every item is packed to exactly one bin. Constraints (1.4) and (1.5) determine that all variables are binary.

BIN PACKING has been shown to be $\mathcal{NP}$–complete in the strong sense (GAREY & JOHNSON, 1979). Various polynomial time approximation algorithms and numerous heuristics have been designed during the last decades. For example, the *First Fit Decreasing (FFD)* algorithm is a simple greedy strategy that places items one after each other in decreasing order of their size into the first bin they fit in. FFD has been shown to use not more than $71/60 \cdot OPT + 1$, where OPT is the optimal solution value (GAREY & JOHNSON, 1985).

There are *Asymptotic Polynomial Time Approximation Schemes (Asymptotic PTAS)* which are able to solve BIN PACKING to any fixed percentage of OPT if the input is sufficiently large (VAZIRANI, 2001). As BIN PACKING is strongly $\mathcal{NP}$-complete, there is no fully polynomial time approximation scheme (FPTAS) as long as $\mathcal{P} \neq \mathcal{NP}$.

However, there are some special cases in which BIN PACKING can be solved to optimality in polynomial time. For example, if all item sizes are divisible by each other – that is for a decreasingly sorted sequence $a_1 > a_2 > \ldots > a_m$ of item sizes a_ℓ, $\ell \in \{1, \ldots, m\}$, it holds that $a_{\ell+1} | a_\ell$ for all $\ell \in \{1, \ldots, m-1\}$ – FFD was shown to produce optimal packings (COFFMAN, JR. ET AL., 1987). If there are only two item sizes with high multiplicities, there is an $\mathcal{O}(\log^2 C)$ algorithm (MCCORMICK ET AL., 2001), where C is the maximum bin capacity. Furthermore, there are polynomial approximation algorithms for the high-multiplicity case with a small number of distinct item sizes (AGNETIS & FILIPPI, 2005).

Innumerable are the applications of BIN PACKING and manifold the techniques that have been applied to the problem over the years. Our individual motivation in writing a thesis in the area of BIN PACKING has multifaceted character: First, we are particularly interested in efficiently solvable special cases of the problem as solutions of provable quality can be obtained within polynomial time. Second, we break out in the direction of a popular programming paradigm named *Constraint Programming*. Within this scope, we are interested in the applicability and algorithmic evaluation of BIN PACKING as a constraint. Third, we are driven by the challenge to efficiently solve large highly–constrained BIN PACKING problems arising from industrial applications.

This thesis illuminates four individual areas of research which are interconnected to each other by the central issue of BIN PACKING. It presents novel theoretical considerations, innovative algorithmic techniques and mathematical models, a wide variety of applications, and experiments in the setting of the BIN PACKING problem.

1.2 Outline and Results

The thesis is divided into 5 independent chapters including the present one. Each chapter is self contained whereas the algorithmic techniques developed in the individual chapters are occasionally used elsewhere. For each chapter, we will give a short outline and summarize the main results.

In Chapter 2, we develop relaxation techniques based on rounding algorithms for arbitrary input data. These techniques might be used either in order to efficiently calculate bounds, or to calculate solutions to the original problem by applying an efficient algorithm to the rounded instance as a special case, or to efficiently detect infeasibility of an instance.

We present several rounding techniques and polynomial algorithms in order to compute an optimal rounding solution. Furthermore, we make qualitative statements on the degree of relaxation in form of proven bounds on the rounding errors and give comprehensive numerical results on their quality on frequently occurring random data profiles and data sets from real-world applications.

In Chapter 3, we consider the high multiplicity variant of the BIN PACKING PROBLEM (BP). That is the case in which only a fixed number m of different object sizes occur in (high) multiplicities. This is of high practical relevance as for example packaging sizes or job lengths in scheduling often fit into this scheme. Moreover, rounding techniques as presented in Chapter 2 might be applied to transform arbitrary BIN PACKING instances into this special case.

We present some ILP models and introduce the notion of dominating lattice points. We prove bounds on the number of these points in lattice polytopes corresponding to (HMBP) instances and give evidence about feasibility of instances. For the case of 2 distinct item sizes, we further improve an algorithm that was developed by AGNETIS & FILIPPI (2005) and MCCORMICK ET AL. (2001). In case of more than 2 distinct item sizes, we present theoretical results that are exploited in a further algorithmic approach. Based on the above considerations, we develop an efficient algorithm for the solution of arbitrary (HMBP) instances which generates solutions that utilize at most one more bin than the optimal solution. Moreover, we show that the size of a solution to an instance of the (HMBP) is polynomial in the size of the input.

In Chapter 4, we develop a framework to jointly evaluate *Bin Packing Constraints* together with *Concave Constraints*. Concave Constraints include all constraints from Mixed Integer Linear Optimization and most logic operations, but are not limited to that: in fact, arbitrary concave functions might be used, for example in order to model functions of growth or decay. Such functions often appear in scheduling, resource allocation or financial problems for example.

The application arises from *Constraint Programming (CP)* – a declarative programming paradigm that states relations between variables in form of constraints. In CP, the initial domains of variables are reduced by means of propagating the set of given constraints on them in order to fix variables to one or more values that satisfy all given constraints. If this has been achieved for all variables, a solution to the constraint program has been found. In Constraint Programming, constraint propagation is often understood as a mere pruning of variable domains. As *Global Constraints* often imply problems from Combinatorial Optimization that are $\mathcal{NP}$-complete, constraint propagation is usually very hard to achieve within an enumerative framework.

In order to evaluate a set of constraints more efficiently, we create an algorithmic framework that is able to detect infeasibility of a set of Bin Packing Constraints and Concave Constraints. We demonstrate that combinations of Bin Packing Constraints and Concave Constraints arise quite naturally in a broad domain of problems from Operations Research and Combinatorial Optimization and, therefore, are particularly promising for practice. We will focus on a selection of $\mathcal{NP}$–hard problems in order to demonstrate that, and in which way, a combination of the Bin Packing Constraint and Concave Constraints may be applied to them. We give special attention to nonlinear constraints. Finally, we present an algorithmic approach that detects infeasibility of a set of *Bin Packing Constraints* and *Concave Constraints*.

In Chapter 5, we consider the *Job Grouping* and *Machine Setup Problem* in automated printed circuit board (PCB) manufacturing. This work arose from a long term cooperation with Philips Assembléon in Eindhoven. In particular, we focus on assembly lines that consist of multiple placement modules and produce several PCB types after each other. The placement modules have feeders that supply a distinct set of component types each. Selected feeders may be exchanged on the fly between the production of two different PCB types by means of changing a trolley (see Figure 1.1). This is called *partly-combined processing*. On the one hand, this allows special component types needed for only few board types in the batch to be set up temporarily. On the other hand, the overall throughput rate

Figure 1.1: Fast component mounter AX-5 featuring 5 exchangeable trolleys each equipped with 4 modules that are able to pick and place electronic components (image by courtesy of Philips Assembléon).

can be increased dramatically by assigning frequently used component types to multiple modules in order to boost the degree of parallelism. As the range of product variants has widened over the years, partly-combined processing is highly relevant for production processes nowadays and in the future.

Trolley exchanges between distinct board types imply a partitioning of the boards into groups. The second problem we face in this context is the creation of feasible machine setups that allow to mount all needed component types efficiently and minimize the total makespan as far as possible. Both problems are closely related to BIN PACKING and might be modeled as a *Multibin Packing Problem*.

We present a comprehensive ILP model of the underlying optimization problem and give a proof of its $\mathcal{NP}$-completeness. Furthermore, we present some theoretic considerations for the solution of the Job Grouping and Machine Setup Problem. Driven by these insights, we develop a novel approach that incorporates setup and load balancing issues already during the grouping phase. The presented approach has many benefits: the computed groupings are feasible and use a minimal number of feeder slots in order to balance workload best possibly between the individual placement modules. Parameter settings leading to infeasible configurations are detected at an early stage of the framework. This results in a much shorter runtime of the optimization software. The computational experiments on large real–world problem data from customers of Philips Assembléon demonstrate that the presented approach is very promising. The quality of solutions computed by our algorithms is vastly superior to those computed by an alternative approach from the literature. Moreover, our approach might be easily customized to a broad range of special hardware requirements of current component mounters.

In the appendix, a glossary of notation that is used in the comprehensive ILP model in Chapter 5 is given for the convenience of the reader. Furthermore, an overview of the notation and an index of keywords used throughout the thesis are given

2 Rounding Techniques for Relaxation

Nicht alles was zählt, kann gezählt werden,
und nicht alles was gezählt werden kann, zählt!

— Albert Einstein

In 1972, Richard Karp showed in his landmark paper 'Reducibility Among Combinatorial Problems' that 21 diverse combinatorial and graph theoretical problems, all infamous for their intractability, were all $\mathcal{NP}$–complete (KARP, 1972). One of them is the KNAPSACK PROBLEM, which can be reduced from EXACT COVER. Nevertheless, there are some special cases of KNAPSACK that can be solved in polynomial time, e.g. if there are only 2 different item sizes (HIRSCHBERG & WONG, 1976) or if all item sizes are divisible (VERHAEGH & AARTS, 1997), i.e. for a decreasingly sorted sequence $a_1 > a_2 > \ldots > a_m$ of item sizes a_ℓ, $\ell \in \{1, \ldots, m\}$, it holds that $a_{\ell+1} | a_\ell$ for all $\ell \in \{1, \ldots, m-1\}$.

For the BIN PACKING PROBLEM, which is $\mathcal{NP}$-complete as well, there are polynomial algorithms in case of divisible item sizes (COFFMAN, JR. ET AL., 1987), or in case of only two item sizes in arbitrary multiplicities (MCCORMICK ET AL., 2001) that solve the problem to optimality. Furthermore, there are fully polynomial approximation algorithms for the high-multiplicity case with a small number of distinct item sizes (AGNETIS & FILIPPI, 2005).

Our idea is to relax the original problem by means of a *rounding technique*. Roughly speaking, several values of the original input are rounded to a common rounding value. Rounding the input corresponds to a surjective transformation, or to a classification of objects into groups of representatives, each containing multiple items. The size of the input thereby shrinks. It is indispensable to minimize the degree of relaxation, that is in our case the *rounding error*, in order to keep negative side effects manageable.

On the one hand, rounding relaxations may be used either explicitly to calculate solutions to the original problem from the relaxation, e.g. by using a solution approach for an efficiently solvable case as mentioned above. On the other hand, rounding relaxations may be used implicitly for the computation of bounds or other certificates of feasibility or infeasibility. Rounding the input is also a widely used technique in order to obtain polynomial time approximation schemes (HOCHBAUM, 1997). For a specific application of rounding techniques to the KNAPSACK PROBLEM see HUTTER & MASTROLILLI (2006).

In the following sections, we will confine ourselves to rounding down values. This means that, for any value a_i from the original data set, the rounding value $\tilde{a}_i$ is strictly lower or equal than the original value a_i. Our motivation to round down the original values arises from the fact that we want to give evidence about the feasibility of decision problems with a *capacity constraint*, also known as *Knapsack constraint*. If the relaxed problem (with values strictly lower or equal than the original values) is infeasible with respect to a capacity constraint, the original problem was infeasible, too. The other way round, if we would round values up, we could deduce from a feasible solution using

relaxed data to a feasible solution of the original instance. All presented algorithms may be adapted straightforwardly to meet the other case.

In this chapter, we will illustrate several rounding techniques that are useful to transform and relax arbitrary input data into structures that might be efficiently used by algorithmic techniques. These are applicable to a wide variety of optimization problems. Arithmetic and geometric rounding are the most successfully and broadly used techniques for the purpose of a relaxations. In addition, we develop a novel rounding technique called *Adaptive Rounding* which outputs a rounding of a very specific structure. This structure might be exploited by subsequent techniques as it is for example done by the relaxations presented in Chapter 4. We will also make quantitative statements on the degree of relaxation in form of bounds on the appropriate rounding errors. Furthermore, we develop efficient algorithms for solving selected rounding problems numerically and give comprehensive computational results on their quality on frequently occurring random data profiles and data sets from real-world applications.

2.1 Rounding of input data

2.1.1 Error Measures

Rounding down the input data of a packing problem amounts to a relaxation of the original problem instance and therefore to an extension of the solution space. Moreover, infeasible problems might become feasible by rounding down original input values. In order to keep any negative side effects that arise from such a relaxation as low as possible, a preprocessing using a rounding technique is usually obliged to minimize the total occurring rounding error. Depending on the problem, the input data and the rounding technique, the following error measures are useful. Without loss of generality, we assume all input and rounding values being positive in this chapter.

Definition 2.1 (Rounding Error)**.** *Without loss of generality, we* round down *every value a_i to not necessarily distinct rounding values $\tilde{a}_i > 0$, for $i \in \{1, \dots, M\}$. Let $\tilde{a}_i > 0$ be the largest rounding value not greater than a_i, i.e. original values a_i are now represented by rounding values $\tilde{a}_i$. For $i \in \{1, \dots, M\}$, we define the* weighted absolute error *by*

$$\underset{A^w}{\Delta} a_i := w_i \cdot (a_i - \tilde{a}_i), \tag{2.1}$$

and the weighted relative error *by*

$$\underset{R^w}{\Delta} a_i := w_i \frac{a_i}{\tilde{a}_i}. \tag{2.2}$$

Both errors are weighted*. In case of using unweighted errors, we will write $\underset{\star}{\Delta}$ instead of $\underset{\star^w}{\Delta}$ and set $w_i := 1$ for all $i \in \{1, \dots, M\}$. For both kinds of errors we consider two objective functions to be minimized, the* maximum error

$$C_{\max} := \max\{\underset{\star^w}{\Delta} a_i \mid i \in \{1, \dots, M\}\}, \tag{2.3}$$

and the sum of errors

$$C_{\Sigma} := \sum_{i=1}^{M} \{\underset{\star^w}{\Delta} a_i\}. \tag{2.4}$$

Some results in the following are valid for all of the above cases. For brevity of notation, we denote the weighted absolute and relative error by

$$\underset{\star^w}{\Delta} a_i := w_i\Big(a_i \ominus \tilde{a}_i\Big), \text{ where } \ominus \in \{-, /\} \text{ and } \star \in \{A, R\}. \tag{2.5}$$

2.1.2 Arithmetic and Geometric Rounding

There are several rounding techniques that are useful for relaxation. The most common ones are *arithmetic rounding* and *geometric rounding* which are used for rounding of linear programs in order to get polynomial time approximation schemes (HOCHBAUM, 1997; HUTTER & MASTROLILLI, 2006).

Definition 2.2 (Generic Arithmetic Rounding Problem)**.** *Given a set of* items $1, \ldots, M$ *with positive* sizes $A = \{a_1, \ldots, a_M\}$*, and a positive integral number* K*. Without loss of generality, we assume that the items are sorted in increasing order. The* Arithmetic Rounding Problem *consists in finding exactly* K *equidistant rounding values* $r_1, \ldots, r_K$ *with* $r_i := r_1 + d \cdot (i-1)$ *so that a given measure of the rounding error is minimized.*

The following proposition is easy to see.

Proposition 2.3. *The arithmetic rounding error is bounded by*

$$\underset{A^w}{\Delta} a_i \quad \leq \quad \max_{j \in \{1,\ldots,M\}} \{w_j\} \cdot \frac{a_M - a_1}{K}, \ \forall i \in \{1, \ldots, M\}, \text{ and} \tag{2.6}$$

$$\underset{R^w}{\Delta} a_i \quad \leq \quad \max_{j \in \{1,\ldots,M\}} \{w_j\} \cdot \big(1 + \frac{a_M - a_1}{K \cdot a_1}\big), \ \forall i \in \{1, \ldots, M\}. \tag{2.7}$$

Definition 2.4 (Generic Geometric Rounding Problem)**.** *Given a set of* items $1, \ldots, M$ *with positive* sizes $A = \{a_1, \ldots, a_M\}$*, sorted increasingly, and a positive integral number* K*. The* Geometric Rounding Problem *consists in finding exactly* K *equidistant rounding values* $r_1, \ldots, r_K$ *with* $r_i := r_1 \cdot d^{i-1}$ *so that a given measure of the rounding error is minimized.*

Proposition 2.5. *There are upper bounds on the geometric rounding error*

$$\underset{A^w}{\Delta} a_i \quad \leq \quad \max_{j \in \{1,\ldots,M\}} \{w_j\} \cdot a_1(d^{K-1} - d^{K-2}), \text{ and} \tag{2.8}$$

$$\underset{R^w}{\Delta} a_i \quad \leq \quad \max_{j \in \{1,\ldots,M\}} \{w_j\} \cdot d \ , \ \forall i \in \{1, \ldots, M\}. \tag{2.9}$$

2.1.3 Rounding to K rounding values

Besides arithmetic and geometric rounding, an interesting rounding technique for relaxation is the rounding to K arbitrary values. This technique addresses principally a special case of optimization problem, that is the case in which only a few values occur in high multiplicities. Naturally, this is the case in the CUTTING STOCK PROBLEM (GILMORE & GOMORY, 1961) and several scheduling problems (HOCHBAUM & SHAMIR, 1991). Clearly, the rounding error is bounded as in the equidistant case by (2.6) and (2.7).

Definition 2.6 (Rounding to K Values Problem)**.** *Given a set of* items $1, \ldots, M$ *with positive* sizes $A = \{a_1, \ldots, a_M\}$*, sorted increasingly, and a positive integral number* K*. The* Rounding to K Values Problem *consists in finding* K *arbitrary rounding values* $r_1, \ldots, r_K$ *in such a way a given rounding error is minimized.*

Algorithm 1 `Rounding to` K `values`

Given: sorted item sizes $a_1, \ldots, a_M$, {absolute|relative} error function $\mathcal{E}$, K
Output: rounding values $r_1, \ldots, r_K$

Initialize:
$lower \leftarrow \mathcal{E}(1, 1)$
$upper \leftarrow$ **maximal Error according to (2.6) resp. (2.7).**
$best \leftarrow \infty$
$thresh \leftarrow$ `bisect`$(lower, upper)$

while (upper – lower > ϵ**) do**
  $max \leftarrow 0$
  $item \leftarrow pos \leftarrow 1$
  while (pos $< K$**) do**
    $feasible \leftarrow$ **FALSE**
    while $(\mathcal{E}(a_{\textbf{item}}, r_{\textbf{pos}}) \leq$ **thresh) do**
      $item \leftarrow item + 1$
    $pos \leftarrow$ **pos+1**
    $r_{\textbf{pos}} \leftarrow item$
    if $(max < \mathcal{E}(a_{\textbf{item}-1}, r_{\textbf{pos}-1}))$ **then**
      $max \leftarrow \mathcal{E}(a_{\textbf{item}-1}, r_{\textbf{pos}-1})$
  if (max $< \mathcal{E}(a_M, r_K))$ **then**
    $max \leftarrow \mathcal{E}(a_M, r_K)$
  if $(max < thresh)$ **then**
    $feasible \leftarrow$ **TRUE**
  if $(max < best)$ **then**
    save this solution as BEST
    $best \leftarrow upper \leftarrow max$
  else if $(feasible)$ **then**
    $upper \leftarrow thresh$
  else $lower \leftarrow thresh$
  $thresh \leftarrow$ `bisect`$(lower, upper)$
return BEST

Algorithm 1 outlines a solution to the problem using binary search in $\mathcal{O}(M \cdot \log E)$, where E is the maximum error from the above mentioned bound. We use an error function $\mathcal{E}$, which simply calculates the rounding error between two values according to Section 2.1.1. Furthermore, we use a function `bisect`, which calculates the mean of two given errors according to the given measure. The algorithm maintains two error bounds: a lower bound stating the minimum error (0 using absolute or 1 using relative errors), and an upper bound from equations (2.6) and (2.7), respectively. Clearly, it constructs a solution that is not worse. Successively, the mean error of the two bounds is calculated. Then, we attempt to construct a solution that does not exceed the given error. If this is successful, we set the upper bound to the currently produced error. If not, we set the lower bound to the mean error. The procedure is repeated with the new bounds until either a given error bound is met, or no further improvement is reached, or the distance between lower and upper bound is less than an $\epsilon > 0$. Clearly, if ϵ is set to machine accuracy, the optimal solution is found upon termination. If we are satisfied with an approximate solution, ϵ determines the quality of approximation.

As the rounding values r_j always coincide with item sizes a_i, the runtime mainly depends on the number of items M.

2.1.4 An Adaptive Rounding Problem

Definition 2.7 (Weighted Adaptive Rounding Problem)**.** *Given a set of* items $\{1,\dots,M\}$ *with positive sizes* $A = \{a_1,\dots,a_M\}$*, a set of* weights $W = \{w_1,\dots,w_M\}$ *and positive integral numbers* K_1 *and* K_2*. The* Weighted Adaptive Rounding Problem *consists in finding exactly* K_1 *intervals with at most* K_2 *equidistant rounding values each with the objective to minimize a given rounding error.*

We denote by the Adaptive Rounding Problem *the unweighted case with all* $w_i \equiv 1$ *for* $i \in \{1,\dots,M\}$.

A feasible solution to the (Weighted) Adaptive Rounding Problem is a quadruple (K, I, X, Δ) with $K \in \mathbb{N}$, $I = (I_0,\dots,I_K)$ with $I_l \in \mathbb{N}, l \in \{0,\dots,K\}$. $X = (X_1,\dots,X_K)$ and $\Delta = (\Delta_1,\dots,\Delta_K)$ with $X_l, \Delta_l \in \mathbb{R}_+$ for $l \in \{1,\dots,K\}$. For $k \in \{1,\dots,K\}$, let

$$m_k := \max\left\{\ell \,\middle|\, \ell \in \mathbb{N};\ \ell \le K_2;\ X_k + \ell\cdot\Delta_k \le a_{I_k}\right\} + 1\,. \tag{2.10}$$

The values $X_k + \ell\cdot\Delta_k$, $k \in \{1,\dots,K\}$, $\ell \in \{0,\dots,m_k-1\}$, are exactly the rounding values $\tilde{a}_i$. For notational convenience, we define $a_0 := 0$. Therefore, a solution (K, I, X, Δ) is feasible if the following conditions hold:

$$K \le K_1, \tag{2.11}$$
$$I_0 = 0,\ I_K = m, \text{ and } I_0 < I_1 < \cdots < I_{K-1} < I_K, \tag{2.12}$$
$$X_0 > 0, \tag{2.13}$$
$$\text{for } k \in \{1,\dots,K\}, \text{ it is } a_{I_{k-1}} < X_k \le a_{I_{k-1}+1}. \tag{2.14}$$

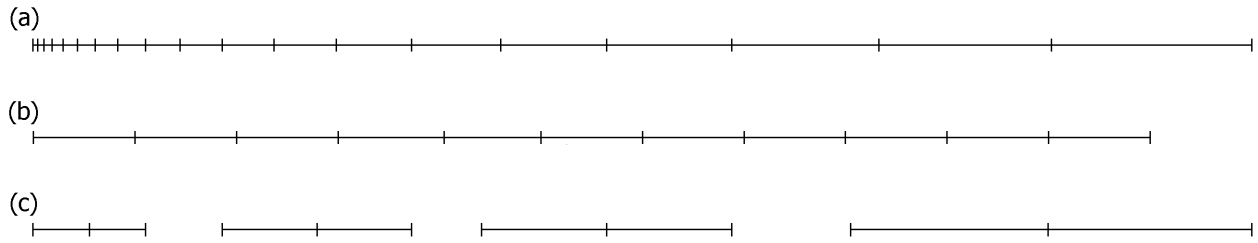


Figure 2.1: Adaptive rounding: (a) log-normal distributed data set of items $a_1,\dots,a_{20}$, (b) equidistant rounding solution $(K_1 = 1, K_2 = 12)$ with $\Delta_{\max} = 13.962$, (c) optimal adaptive rounding solution for $K_1 = 4, K_2 = 3$ with $\Delta_{\max} = 3.322$ (maximum of absolute errors)

The first condition (2.11) states that the number K of intervals is at most K_1. Condition (2.12) says that the set of items $\{1,\dots,M\}$ is partitioned into intervals $[I_{i-1}+1,\dots,I_i]$, $i \in \{1,\dots,M\}$. (2.13) guarantees everything is non-negative, and finally, (2.14) ensures that the left–hand borderline of each interval of equidistant rounding values is not right of any value a_i to be rounded down to one of these rounding values.

The rounding error is defined as in Definition 2.1. We choose

$$\tilde{a}_i := \max\left\{ X_{h_i} + \ell\cdot\Delta_{h_i} \,\middle|\, \ell \in \mathbb{Z},\ X_{h_i} + \ell\cdot\Delta_{h_i} \le a_i \right\}, \tag{2.15}$$

that is, $\tilde{a}_i$ is the largest rounding value not greater than a_i, for all $i \in \{1, \ldots, M\}$. h_i denotes the value $k \in \{1, \ldots, K\}$ such that $I_{k-1} < i \leq I_k$.

Figure 2.1 shows log-normally distributed data in the interval $[0; 200]$ and two rounding solutions: one using rounding to 12 equidistant rounding values, and one using Adaptive Rounding to 4 intervals of 3 rounding values each. The maximum over all absolute rounding errors is 13.962 in the equidistant solution versus 3.322 in the optimal Adaptive Rounding solution.

2.1.5 Bounds on the Rounding Error

For lucidity in the forthcoming sections, we first confine ourselves to treating the problem using error maxima. Section 2.2.5 then contains the modifications to be done for using error sums.

Theorem 2.8. *Consider an optimal solution to the rounding problem. Then for $i, j \in \{1, \ldots, M\}$, there are upper bounds on*

(a) the absolute error: $\underset{A^w}{\Delta} a_i \leq \max_j\{w_j\} \cdot \dfrac{a_M - a_1}{K_1 \cdot K_2}$, *and*

(b) the relative error: $\underset{R^w}{\Delta} a_i \leq \max_j\{w_j\} \cdot \left(1 - \dfrac{1}{K_2} + \dfrac{1}{K_2}\left(\dfrac{a_M}{a_1}\right)^{1/K_1}\right)$.

Proof. We will prove Theorem 2.8 for the unweighted case by constructing two feasible solutions (K, I, X, Δ), each of which fulfills one of these inequalities. Clearly, the optimal solution cannot be worse. The weighted case can be proven analoguesly.
(a) Absolute error: For the absolute error, it is easy to see that the error is bounded by the maximum error arising from a solution with the same number of equidistant rounding values. Therefore we choose $K_1 \cdot K_2$ equidistant rounding values over the interval $\left[a_1; a_M\right]$ and $\frac{a_M - a_1}{K_1 \cdot K_2}$ as the distance between those values. Clearly, the maximum error could not exceed this bound. Theorem 2.8a follows immediately for the weighted absolute error.
(b) Relative error: For the construction of a feasible solution, we define values $X_1, \ldots, X_{K_1} \in [a_1 \ldots a_M]$ and define a ratio $Q := X_{i+1}/X_i$ for all $i \in \{1, \ldots, K_1\}$. Q is then uniquely determined:

$$Q = \left(\frac{a_M}{a_1}\right)^{1/K_1}.$$

Finally, for $k \in \{1, \ldots, K_1\}$, we set $\Delta_i := (X_{k+1} - X_k)/K_2$.

Consider an arbitrary $i \in \{1, \ldots, M\}$, and let $k_1 \in \{1, \ldots, K_1\}$ and $k_2 \in \{0, \ldots, K_2 - 1\}$ be defined such that $\tilde{a}_i = X_{k_1} + k_2 \cdot \Delta_{k_1}$. Then we have $a_i \leq X_{k_1} + (k_2 + 1) \cdot \Delta_{k_1}$. Substituting $\Delta_{k_1} = (X_{k_1+1} - X_{k_1})/K_2$ yields

$$\underset{R}{\Delta} a_i = \frac{a_i}{\tilde{a}_i} \leq \frac{X_{k_1} + (k_2 + 1) \cdot \Delta_{k_1}}{X_{k_1} + k_2 \cdot \Delta_{k_1}} = \frac{X_{k_1} + \frac{k_2+1}{K_2} \cdot (X_{k_1+1} - X_{k_1})}{X_{k_1} + \frac{k_2}{K_2} \cdot (X_{k_1+1} - X_{k_1})}.$$

If k_2 is variable and everything else is fixed, the last expression assumes its maximum at $k_2 = 0$, where it simplifies to

$$\underset{R}{\Delta} a_i = \frac{X_{k_1} + \frac{1}{K_2} \cdot (X_{k_1+1} - X_{k_1})}{X_{k_1}} = 1 - \frac{1}{K_2} + \frac{Q}{K_2}.$$

The weighted case of Theorem 2.8b follows immediately. □

Therefore, we may say that K_1 and K_2 specify the degree of discretization. The greater the values for K_1 and K_2, respectively, are chosen, the more classes of items will exist, but the smaller the rounding error gets. Hence, we have a tradeoff between the number of item classes, and thus the computation speed and the discretization error.

2.2 An Efficient Solution to the Adaptive Rounding Problem

2.2.1 Discretization

The following lemma shows that we may restrict the search for optimal solutions to the Adaptation Rounding Problem to a discrete set of size $O(M^2)$.

Lemma 2.9. *There is an optimal solution (K, I, X, Δ) such that one of the following conditions is fulfilled for every $k \in \{1, \ldots, K\}$:*

1. *If $m_k = 1$, there is $i \in \{1, \ldots, M\}$ such that $h_i = k$ and $a_i = X_k$.*
2. *On the other hand, if $m_k > 1$, one of the following two conditions is fulfilled:*
   a) *"2-point fit": There are $i_1, i_2 \in \{1, \ldots, M\}$ and $\ell_1, \ell_2 \in \{0, \ldots, m_k - 1\}$ such that $h_{i_1} = h_{i_2} = k$, $a_{i_1} = X_k + \ell_1 \cdot \Delta_k$, and $a_{i_2} = X_k + \ell_2 \cdot \Delta_k$.*
   b) *"equilibrium": There are $i_1, i_2, i_3 \in \{1, \ldots, M\}$ and $\ell_2 \in \{0, \ldots, m_k - 1\}$ such that $h_{i_1} = h_{i_2} = h_{i_3} = k$, $i_1 < i_2 < i_3$, $a_{i_2} = X_k + \ell_2 \cdot \Delta_k$, and the maximum*

$$\max \left\{ w_i \Big(a_i \ominus \tilde{a}_i \Big) \,\Big|\, i \in \{1, \ldots, M\},\, h_i = k \right\},\ \ominus \in \{-, /\},$$

   is attained at both indices, $i = i_1$ and $i = i_3$.

Proof. Consider an optimal solution (K, I, X, Δ), and let $k \in \{1, \ldots, M\}$. First suppose that $a_i \neq X_k + \ell \cdot \Delta_k$ for all $i \in \{1, \ldots, M\}$ and $\ell \in \{0, \ldots, m_k - 1\}$ where $h_i = k$. Let

$$\varepsilon := \min \left\{ a_i - \tilde{a}_i \,\middle|\, i \in \{1, \ldots, M\};\, h_i = k \right\} > 0\,.$$

Further, let X' be the K–vector defined by $X'_k := X_k + \varepsilon$ and $X'_j := X_j$ for all $j \in \{1, \ldots, K\}$, $j \neq k$. Obviously, (K, I, X', Δ) is better than (K, I, X, Δ) with respect to the objective function. This proves the theorem for case 1 completely, and for case 2 partially, namely, the existence of i_2 and ℓ_2 is already proven. To finish the proof of the theorem for case 2, assume that $m_k > 1$, and that the scenario of case 2a is not fulfilled. We have to show that then case 2b is fulfilled.

Then i_2 is the only index such that a_{i_2} is one of the rounding values. Therefore, the maximum $\max\{a_i - \tilde{a}_i \,|\, i \in \{1, \ldots, M\},\, h_i = k\}$ cannot be assumed at $i = i_2$, because then this maximum is 0, and thus each a_i would be a rounding value. This would contradict the assumption $m_k > 1$. In summary, the maximum is attained at some $i_1 < i_2$ or at some $i_3 > i_2$.

It remains to show that the maximum is assumed at some $i_1 < i_2$ and, simultaneously, at some $i_3 > i_1$. So suppose for a contradiction that the maximum is only assumed at indices $i_1 < i_2$ or only at indices $i_3 > i_2$. For $\varepsilon \in \mathbb{R}$, let $(K, I, X_\varepsilon, \Delta_\varepsilon)$ be defined by

- $X^\varepsilon_j := X_j$ and $\Delta^\varepsilon_j := \Delta_j$ for all $j \in \{1, \ldots, K\} \setminus \{k\}$;
- $\Delta^\varepsilon_k := \Delta_k + \varepsilon$;
- $X^\varepsilon_k := a_{i_2} - \ell_2 \cdot \Delta^\varepsilon_k$.

Now we can express $\tilde{a}^\varepsilon_i$ as a linear function of ε for each item i with $h_i = k$:

$$\begin{aligned}
\tilde{a}^\varepsilon_i &= X^\varepsilon_k + \ell \Delta^\varepsilon_k \\
&= a_{i_2} - \ell_2 \Delta^\varepsilon_k + \ell \Delta^\varepsilon_k \\
&= X_k + \ell \Delta^\varepsilon_k - (\ell_2 - \ell) \Delta^\varepsilon_k \\
&= X_k - (\ell_2 - \ell)\varepsilon + \ell \Delta_k \\
&= \tilde{a}_i - (\ell_2 - \ell)\varepsilon.
\end{aligned}$$

Hence, for sufficiently small $\varepsilon > 0$, we have the case that $\tilde{a}_i^\varepsilon$ is monotonously increasing in $\varepsilon > 0$ if $\ell > \ell_2$ and decreasing, otherwise. In other words, $w_i(a_i \ominus \tilde{a}_i^\varepsilon)$ is monotonously decreasing if $\ell > \ell_2$ and increasing, otherwise. The converse relations hold if we choose a small negative ε. Therefore, with the appropriate choice of ε we can strictly decrease the objective function value whenever the maximum is different for values $i < i_2$ and $i > i_2$, respectively. This contradiction eventually proves Lemma 2.9. □

2.2.2 Overview of the Algorithm

Based on the the results above, we introduce an efficient algorithm for the Adaptive Rounding Problem.

Theorem 2.10. *For K_1 and K_2 fixed, there is an $\mathcal{O}(M^2)$ algorithm for the Adaptive Rounding Problem.*

Sections 2.2.2 until 2.2.4 are devoted to the proof of Theorem 2.10.

In the following, we consider a restricted version of the problem, namely restricted to the special case $K_1 = 1$. This is tantamount to rounding down arbitrary values to a batch of K_2 equidistant values. To distinguish solutions to the restricted version from solutions to the general version, we will not denote a solution to the restricted version by $(1, I, X, \Delta)$. Instead, we will denote such a solution by (x, δ, k). This means that the rounding values are x, $x + \delta$, $x + 2 \cdot \delta$, $\ldots, x + (k-1) \cdot \delta$.

The algorithm consists of two stages. The idea is that a feasible solution (K, I, X, Δ) may be viewed as K quadruples $(1, I_j, X_j, \Delta_j)$, $j \in \{1, \ldots, K\}$, where I_j is the tuple $I_j = (I_{j-1}, I_j)$. Such a quadruple $(1, I_j, X_j, \Delta_j)$ is feasible for a subinterval $\{a_{i_1}, \ldots, a_{i_2}\}$ of $\{a_1, \ldots, a_M\}$ if, and only if, $a_{i_1} \geq X_j$. In particular, it is feasible for $i_1 \geq I_{j-1} + 1$.

Let (K, I, X, Δ) be an optimal solution to the adaptive rounding problem for $\{a_1, \ldots, a_M\}$. Moreover, for $j \in \{1, \ldots, K\}$ let k_j be the maximal integral number such that $k_j < K_2$ and $X_j + k_j \cdot \Delta_j \leq a_{I_j}$. Obviously, (X_j, Δ_j, k_j) is then an optimal solution to the restricted Adaptive Rounding Problem for the subinterval $\{a_{I_{j-1}+1}, \ldots, a_{I_j}\}$. From the problem definition we know that $a_{I_{j_0}} < X_j \leq a_{I_{j_0}+1}$. Therefore, in the following we will restrict ourselves to triples (x, δ, k) such that $a_{I_{j_0}} < x \leq a_{I_{j_0}+1}$.

In Stage 2, (K, I, X, Δ) is then composed of triples (x, δ, k) of this kind in a dynamic–programming scheme. For that, Stage 1 computes in a sweep-line fashion, all triples (x, δ, k) such that (x, δ, k) may be potentially relevant for Stage 2. The optimal triples (x, δ, k) for all subintervals of $\{a_1, \ldots, a_M\}$ in the restricted problem are saved. Lemma 2.9 will be utilized in Stage 1. In fact, this lemma says that we may restrict our focus on triples (x, δ, k) such that

- $a_i = x + \ell \cdot \delta$ for at least one pair $i \in \{1, \ldots, M\}$ and $\ell \in \{0, \ldots, k-1\}$ and
- at least one of the following two conditions is fulfilled:
  (1) $a_j = x + \mu \cdot \delta$ for some other pair $j \in \{1, \ldots, M\} \setminus \{i\}$ and $\mu \in \{0, \ldots, k-1\} \setminus \{\ell\}$ or
  (2) the maximum of $w_j(a_j \ominus \tilde{a}_j)$ is assumed for some j_1, $j_2 \in \{1, \ldots, M\}$ such that $j_1 < i < j_2$ (equilibrium situation).

Therefore, in Stage 1 all triples (x, δ, k) that fulfill these conditions are systematically enumerated. We will see that the number of triples to be computed is $\mathcal{O}(M^2)$ for fixed K_2, and that the computation of each triple takes amortized constant time. From the problem definition we know it suffices to consider a triple (x, δ, k) solely for subintervals $\{a_{i_1}, \ldots, a_{i_2}\}$ such that $a_{i_1-1} < x \leq a_{i_1}$. Therefore, we do not need to save all calculated triples, but only the best for the appropriate subinterval.

In summary, these are the two stages of the algorithm:

- **Stage 1: Generating relevant solutions for the restricted problem**
  All triples (x, δ, k) that fulfill the conditions of Lemma 2.9 are calculated. Moreover, for each of these triples, additional information about the objective function and the subinterval $[a_i, a_j]$ the triple is valid for is simultaneously computed.

  The triples are saved in a solution matrix B of dimension $M \times M$. The matrix B is triangular; it contains a triple to each subinterval $[a_i, \ldots, a_j]$ such that $i \leq j$. Whenever a triple is found for $[a_i, \ldots, a_j]$ that is better than the triple currently stored as the entry b_{ij}, b_{ij} is overwritten by the new triple.

  In a post-processing step of Stage 1, we have to determine, whether the objective function value of the triple stored at b_{ij} is better than that of the triples at the positions b_{ij-1}, $b_{ij-2}, \ldots$. On the other hand, we also check the positions b_{ij+1}, $b_{ij+2} \ldots$, to determine whether the triple stored at b_{ij} extended to one of those larger subintervals is even better than the triple stored for that subinterval. At last, entries at b_{i+1j}, $b_{i+2j} \ldots$ are checked.

- **Stage 2: Getting optimal solution(s) to the original problem**
  An optimal solution to the original problem is composed of these triples using a dynamic programming approach. Possibly, more than one solution with the optimal objective function value exist. In this case, we can either save all of them, or keep just one.

  The output of Stage 2 consists of an objective function value (the calculated error), at most K_1 triples (x, δ, k) and the information, how many item values have been rounded down to the corresponding rounding values. The first two components of the k–th triple are then the values X_k and Δ_k of the final solution (and the third component of the k–th triple is the associated value m_k).

2.2.3 Stage 1: Relevant Solutions for the Restricted Problem

In Stage 1, for each $i \in \{1, \ldots, M\}$, $k \in \{1, \ldots, K_2 - 1\}$, and $\ell \in \{1, \ldots, k\}$, the set of triples for (i, k, ℓ) is computed by a loop in a sweep–line like fashion. The μ–th iteration of this loop computes one triple (x_μ, δ_μ, k) and its objective function value. To get the loop started, an initial triple (x_0, δ_0, k) is computed in a preprocessing step, which is performed before each sweep–line pass. It will be obvious from the procedure that $\delta_0 < \delta_1 < \delta_2 < \cdots$.

To realize this loop correctly, we have to ensure that no potentially relevant δ–value is in between $\delta_{\mu-1}$ and δ_μ for any $\mu > 0$. Based on Lemma 2.9, this is ensured as follows. First, we set $\delta_0 := \min\{a_i - a_{i-1}, a_{i+1} - a_i\}$ for $i \in \{2, \ldots, M-1\}$. For $i = 1$, we set $\delta_0 := a_2 - a_1$, for $i = M$, we set $\delta_0 := a_M - a_{M-1}$. Clearly, no smaller δ–value may be relevant.

The crucial observation is that the rounding values x_μ, $x_\mu + \delta_\mu$, $x_\mu + 2 \cdot \delta_\mu$, $\ldots$, $x_\mu + (k-1) \cdot \delta_\mu$ partition $\{a_i, \ldots, a_j\}$ with $a_i \geq x$ and $a_j < x + k\delta$ into $k - 1$ (at most K_2) linearly ordered subsets. Possibly some of them are empty. Roughly speaking, two items belong to the same subset with respect to μ if they are in the same interval between two successive rounding values, $x_\mu + \lambda \cdot \delta_\mu$ and $x_\mu + (\lambda + 1) \cdot \delta_\mu$, $\lambda \in \{0, \ldots, k-1\}$. The open point in this informal definition is where to put items a_j that are on a "borderline" between two buckets, that is $a_j = x_\mu + \lambda \cdot \delta_\mu$ for some λ. To motivate our strategy for these items intuitively, imagine that the λ–th rounding value is a sweep value, which sweeps over the items as δ increases. Solely the ℓ–th rounding value is fixed at a_i.

As the items a_j are ordered, a partition into the buckets is most practicable. We want to have items a_{j_1} and a_{j_2} in a common bucket if they are rounded down to or are identical with the next

rounding value. The partitioning is completely described by maintaining for each bucket the index of the biggest item which it belongs to. This is done by the discrete mapping function

$$p_\mu : \mathbb{N}_0 \to \mathbb{N}_0,\ \lambda \mapsto \max\{j \mid a_j < x+\lambda\delta\}. \tag{2.16}$$

Hence, for a given $\mu \geq 0$, we denote these indices by $p_\mu(0), p_\mu(1), \ldots, p_\mu(k) \in \{0, \ldots, M\}$, $p_\mu(0) \leq p_\mu(1) \leq \cdots \leq p_\mu(k)$. In other words, the λ–th bucket after μ iterations of the loop ($\mu = 0, 1, 2, \ldots$) is then $a_{p_\mu(\lambda-1)+1}, \ldots, a_{p_\mu(\lambda)}$. The above definition of buckets (in the following we will call these buckets the *inner buckets*) is enhanced by two more buckets for items that are not rounded down to specific rounding values: a *lower* and an *upper outer bucket*.

Definition 2.11. *During Stage* 1 *of the Adaptive Rounding Problem Algorithm, the following* buckets *are defined:*

1. inner buckets:

   For all $\lambda \in \{1, \ldots, k-1\}$ *and for all* $j \neq i$*, it is* $j \in \{p_\mu(\lambda-1)+1, \ldots, p_\mu(\lambda)\}$ *if, and only if,* $x_\mu + (\lambda-1)\cdot\delta_\mu \leq a_j < x_\mu + \lambda\cdot\delta$.

2. outer buckets:

   a) lower outer bucket (items below x_μ)
   For $\lambda = 0$*, it is* $j \in \{1, \ldots, p_\mu[0]\}$ *if, and only if,* $0 < a_j < x$.

   b) upper outer bucket (items above $x_\mu + (k-1)\cdot\delta_\mu$)
   For $\lambda = k$*, it is* $j \in \{p_\mu(k-1)+1, \ldots, p_\mu(k)\}$ *if, and only if,* $x_\mu + (k-1)\cdot\delta_\mu \leq a_j$.

Figure 2.2 illustrates the definition of buckets by means of an example.

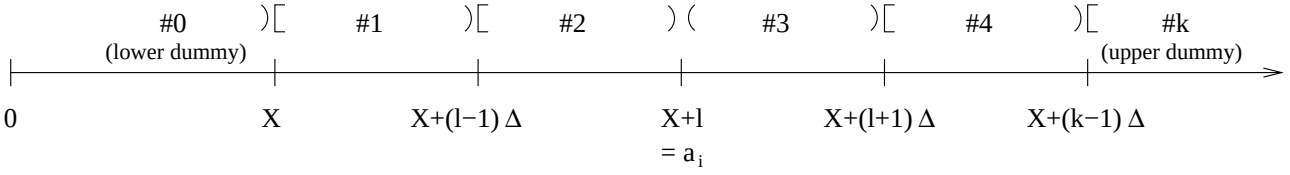


Figure 2.2: Definition of buckets according to Definition 2.11 for $k = 5$.

In the μ–th iteration, we first compute δ_μ. Basically, we have to ensure that no δ–value between $\delta_{\mu-1}$ and δ_μ is potentially relevant. We can utilize Lemma 2.9 to determine how large the increment κ in the update formula

$$\delta_\mu = \delta_{\mu-1} + \kappa \tag{2.17}$$

may be at most. More specifically, we set

$$\kappa_\mu(\lambda) > 0,\ \kappa_\mu(\lambda) := \begin{cases} \frac{X+\lambda\Delta-a_{p_\mu(\lambda)}}{\ell-\lambda},\ \lambda < \ell, \\ \frac{X+\lambda\Delta-a_{p_\mu(\lambda)+1}}{\ell-\lambda},\ \lambda > \ell, \\ +\infty,\ \lambda = \ell. \end{cases} \tag{2.18}$$

That is, $\kappa_\mu(\lambda)$ is the minimal value by which $\delta_{\mu-1}$ must be increased such that the rounding value $x_\mu + \lambda\delta_\mu$ sweeps over one item in the λ–th bucket. Finally, we set

$$\kappa_\mu = \min\left\{\kappa_\mu(\lambda) \mid \lambda \in \{1,\ldots,k\},\ \kappa_\mu(\lambda) > 0\right\}. \tag{2.19}$$

In other words, κ_μ is the minimal value by which $\delta_{\mu-1}$ must be increased such that at least one rounding value – namely that one which is closest to an item (in a sense to be specified below) – sweeps over that item in the λ–th bucket. Clearly, this minimum can be assumed at more than one rounding value. If there is no such increment $\kappa_\mu(\lambda)$, which permanently happens in the case $\lambda = \ell$, then we set $\delta(\lambda) := +\infty$.

Furthermore we essentially have to regard the case

$$w_{p_\mu(\lambda_1)} \cdot \left(a_{p_\mu(\lambda_1)} - (x_\mu + \lambda_1\delta_\mu)\right) = \left(a_{p_\mu(\lambda_2)} - (x_\mu + \lambda_2\delta_\mu)\right) \cdot w_{p_\mu(\lambda_2)}, \tag{2.20}$$

applying the absolute error, and

$$w_{p_\mu(\lambda_1)} \cdot \frac{a_{p_\mu(\lambda_1)}}{x_\mu + \lambda_1\delta_\mu} = \frac{a_{p_\mu(\lambda_2)}}{x_\mu + \lambda_2\delta_\mu} \cdot w_{p_\mu(\lambda_2)}, \tag{2.21}$$

applying the relative error for each pair (λ_1, λ_2) with $\lambda_1 \leq \ell$ and $\lambda_2 > \ell$. In this case two rounding errors of the same size emerge in different buckets, one on the left and one on the right hand side of the fixed rounding value a_i. These rounding errors reflect the maximum rounding error in the related subinterval. If now Δ is increased, the rounding error on the right hand side is decreasing while that on the left hand side is growing. We will call this situation *equilibrium* in the following context. This equilibrium situation is depicted in Figure 2.3.

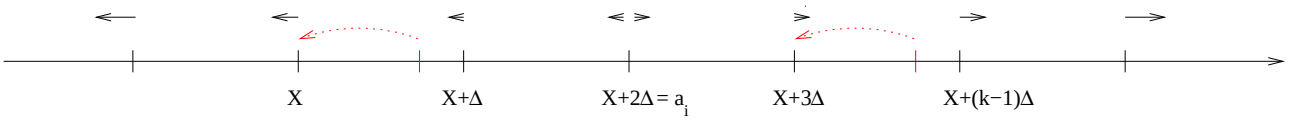


Figure 2.3: Equilibrium situation using absolute errors (cf. equation (2.20)).

During the search for a minimal κ we have to take into account that a minimal κ may lead to the above described situation. Therefore, in case of absolute errors, we have to check whether

$$\begin{aligned} & w_{p_\mu(\lambda_1)} \cdot \left(a_{p_\mu(\lambda_1)} - (x_{\mu-1} + \lambda_1(\delta_{\mu-1} + \kappa_\mu(\lambda_1,\lambda_2)))\right) \\ = {} & w_{p_\mu(\lambda_2)} \cdot \left(a_{p_\mu(\lambda_2)} - (x_{\mu-1} + \lambda_2(\delta_{\mu-1} + \kappa_\mu(\lambda_1,\lambda_2)))\right), \end{aligned} \tag{2.22}$$

and in case of relative errors, whether

$$\frac{w_{p_\mu(\lambda_1)} \cdot a_{p_\mu(\lambda_1)}}{x_{\mu-1} + \lambda_1(\delta_{\mu-1} + \kappa_\mu(\lambda_1,\lambda_2))} = \frac{w_{p(\lambda_2)} \cdot a_{p_\mu(\lambda_2)}}{x_{\mu-1} + \lambda_2(\delta_{\mu-1} + \kappa_\mu(\lambda_1,\lambda_2))}. \tag{2.23}$$

Again, this has to be done for each pair (λ_1, λ_2) with $\lambda_1 \leq \ell$ and $\lambda_2 > \ell$. Therefore, in the case of absolute values, we get additional values

$$\kappa_\mu(\lambda_1,\lambda_2) := \frac{w_{p(\lambda_2)} \cdot (a_{p_\mu(\lambda_2)} - x_{\mu-1}) - w_{p(\lambda_1)} \cdot (a_{p_\mu(\lambda_1)} - x_{\mu-1})}{w_{p(\lambda_2)}\lambda_2 - w_{p(\lambda_1)}\lambda_1} - \delta, \tag{2.24}$$

and in the case of relative errors, we get additional values

$$\kappa_\mu(\lambda_1,\lambda_2) \;:=\; \frac{(w_{p(\lambda_1)}a_{p_\mu(\lambda_1)} - w_{p(\lambda_2)}a_{p_\mu(\lambda_2)})\cdot x}{w_{p(\lambda_2)}a_{p_\mu(\lambda_2)}\lambda_1 - w_{p(\lambda_1)}a_{p_\mu(\lambda_1)}\lambda_2} - \delta\,, \tag{2.25}$$

and define the minimum of them as

$$\kappa_{\mu\mu} := \min\Big\{\kappa_\mu(\lambda_1,\lambda_2) \mid \lambda_1,\lambda_2 \in \{1,\ldots,k\},\ \lambda_1 \le \ell < \lambda_2,\ \kappa_\mu(\lambda_1,\lambda_2) > 0\Big\}. \tag{2.26}$$

Thus, the increment κ in the update formula (2.17) is set to

$$\kappa \;:=\; \min\Big\{\kappa_\mu, \kappa_{\mu\mu}\Big\}. \tag{2.27}$$

In this way, we get as a result a kind of sweep–line algorithm, which guarantees that no potentially relevant δ–value is omitted. The sweep-line loop terminates when $\kappa = \infty$. So suppose $0 < \kappa < \infty$ in the following.

After the δ–update, we set $x_\mu := a_i - \ell\cdot\delta_\mu$. Whenever the minimum κ is assumed at some $\kappa_\mu(\lambda)$, we also have to update the bucket structure. Namely, we set $p_\mu(\lambda) := p_{\mu-1}(\lambda) - 1$ in case $k < \ell$, and $p_\mu(\lambda-1) := p_{\mu-1}(\lambda-1) + 1$ in case $k > \ell$. All other values $p_\mu(\lambda)$ are defined as $p_\mu(\lambda) := p_{\mu-1}(\lambda)$. Obviously, for $\lambda = \ell$ and $\lambda = \ell+1$, we always have $p_\mu(\lambda) = p_{\mu-1}(\lambda)$.

Given the partition into buckets, each solution (x_μ,δ_μ,k), induces a cost value

$$C_\mu := \max\left\{ w_i\big(a_i \ominus \tilde{a}_i\big) \;\middle|\; i\in\{1,\ldots,M\}, x_\mu \le a_i \le x_\mu + k\cdot\delta_\mu \right\}. \tag{2.28}$$

Namely, this is the maximum absolute rounding error over all non-empty inner buckets. In the unweighted case, the maximum error C_μ can be determined as the maximum of $k-1$ values, one for each bucket, as the maximum ratio within each bucket is assumed for the largest element. In the weighted case, all items in the inner buckets have to be considered.

Given C_μ and the largest rounding value $x_\mu + k\cdot\delta_\mu$ for the triple (x_μ,δ_μ,k), we can compute the point $x_{\max} := C_\mu + (x_\mu + k\cdot\delta_\mu)$ in the absolute and $x_{\max} := (C_\mu+1)(x_\mu + k\cdot\delta_\mu)$ in the relative case. The interpretation of this point is the following: For items from the last bucket that satisfy $w_i a_i > x_{\max}$, the rounding error $w_i\big(a_i \ominus \tilde{a}_i\big)$ will be strictly larger than C_μ. We define j_μ as one index below the first index in the last bucket for that $w_i a_i > x_{\max}$. As the objective function right hand side of $x_{\max}$ is piecewise-linear, but not necessarily monotonous due to the weights w_i, we have to pass through the indices of the last bucket in increasing order. In the unweighted case, the objective function for the triple (x_μ,δ_μ,k) is monotonously increasing and linear in the size a_j of the last item j if $a_j \ge x_{\max}$. Figure 2.4 illustrates the definition of $C_\mu, x_{\max}$ and j_μ.

We also define $i_\mu \in \{1,\ldots,M\}$ such that $a_{i_\mu-1} < x_\mu \le a_{i_\mu}$ in case $x_\mu \ge a_1$ and $i_\mu = 1$ otherwise. The restricted item set the solution is valid for is then $\{i_\mu,\ldots,j_\mu\}$ for $i_\mu < j_\mu \le M$.
We maintain a solution matrix $\boldsymbol{B} := (b_{ij})$, $\boldsymbol{B} \in M \times M$. Once a triple (x_μ,δ_μ,k) has been calculated, i_μ and j_μ are determined. The triple overwrites the entry b_{i_μ,j_μ} in the solution matrix $\boldsymbol{B}$ if its objective function value is better than that of the triple currently stored at b_{i_μ,j_μ}. Otherwise, it is discarded, and $\boldsymbol{B}$ remains unchanged.

Furthermore, we have to regard solutions at $b_{i_\mu-\nu,j_\mu}$ for decreasing ν as long as the objective function value of $b_{i_\mu-\nu,j_\mu}$ is greater than that at b_{i_μ,j_μ}. On the other hand, we have to investigate solutions at $b_{i_\mu+\nu,j_\mu}$ for increasing ν, to ensure that there is a better solution yet. If not, we save the current solution (x_μ,δ_μ,k) with the limits $\{i_\mu,\ldots,j_\mu+\nu\}$ and the objective function value $a_{j_\mu+\nu} - (x_\mu + k\delta_\mu)$. This is usually done once after completion of Stage 1 in a post-processing step for the whole matrix $\boldsymbol{B}$.

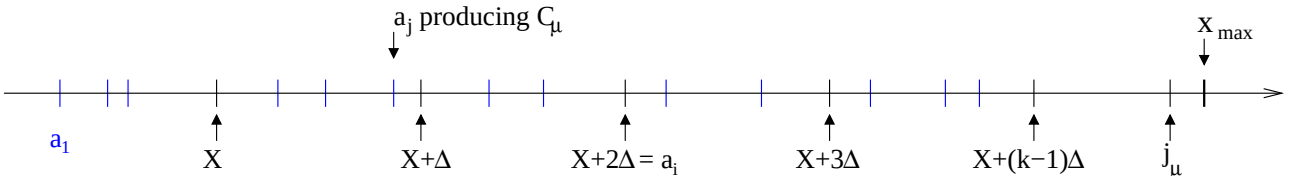


Figure 2.4: Calculation of C_μ, $x_{\max}$ and j_μ using absolute errors.

Lemma 2.12. *For K_2 fixed, the first major step of the algorithm requires $\mathcal{O}(M^2)$ asymptotic run time.*

Proof. We will prove Lemma 2.12 applying absolute errors. The proof for the relative case acts exactly analogous.

It is easy to see that, for fixed K_2, the run time is linear in the number of generated pairs (x_μ, δ_μ). Therefore, it suffices to show that $\mathcal{O}(M)$ pairs (x_μ, δ_μ) are generated for each combination $i \in \{1, \ldots, M\}$, $\ell \in \{0, \ldots, K_2 - 1\}$. Let $i \in \{1, \ldots, M\}$ and $\ell \in \{1, \ldots, K_2\}$ be fixed in the following. Thus, we concentrate on the pairs (x, δ) where $a_i = x + \ell \cdot \delta$. In principle, two kinds of pairs are generated:

- pairs (x, δ) where $a_j = x + r \cdot \delta$ for some $j \in \{1, \ldots, M\} \setminus \{i\}$ and $r \in \{1, \ldots, K_2\} \setminus \{\ell\}$,
- pairs $(x, \delta)$where the objective function value is

$$w_{j_2} \cdot \big(a_{j_1} - x + r_1 \cdot \delta\big) \quad = \quad \big(a_{j_2} - x + r_2 \cdot \delta\big) \cdot w_{j_1}$$

  for some $j_1, j_2 \in \{1, \ldots, M\}$, $j_1 < i < j_2$, and $r_1, r_2 \in \{1, \ldots, K_2\}$, in particular $r_1 < \ell < r_2$ (equilibrium situation).

Obviously, there are at most $M \cdot K_2$ pairs of the first kind, because the conditions $x_\mu + \ell \cdot \delta_\mu = a_i$ and $x_\mu + r \cdot \delta = a_j$ determine μ uniquely. To prove the lemma, it thus suffices to show that, for each pair (x_μ, δ_μ) of the second type, the very next pair $(x_{\mu+1}, \delta_{\mu+1})$ is of the first type.

To see this, consider the μ–th iteration of the loop, and suppose for a contradiction that both the μ–th and the $(\mu + 1)$–st pair are of the second type. Let j_1, j_2, r_1, and r_2 be as defined above for (x_μ, δ_μ) and j_1', j_2', r_1', and r_2' for $(x_{\mu+1}, \delta_{\mu+1})$. This implies

$$a - j_1 - (x_\mu + r_1 \cdot \delta_\mu) \;\geq\; a - j_1' - (x_\mu + r_1' \cdot \delta_\mu), \quad \text{and} \tag{2.29}$$

$$a_{j_2} - (x_{\mu+1} + r_2 \cdot \delta_{\mu+1}) \;\leq\; a_{j_2'} - (x_{\mu+1} + r_2' \cdot \delta_{\mu+1})\,. \tag{2.30}$$

Since $r_1' < \ell$, we have $x_{\mu+1} + r_1' \cdot \delta_{\mu+1} < x_\mu + r_1' \cdot \delta_\mu$. On the other hand, $r_2 > \ell$ implies $x_{\mu+1} + r_2 \cdot \delta_{\mu+1} > x_\mu + r_2 \cdot \delta_\mu$. Altogether, we obtain the following contradiction:

$$\begin{aligned} &a_{j_1} - (x_\mu + r_1 \cdot \delta_\mu) \geq a - j_1' - (x_\mu + r_1' \cdot \delta_\mu) \;>\; a_{j_1'} - (x_{\mu+1} + r_1' \cdot \delta_{\mu+1}) \\ &= a_{j_2'} - (x_{\mu+1} + r_2' \cdot \delta_{\mu+1}) \geq a_{j_2} - (x_{\mu+1} + r_2 \cdot \delta_{\mu+1}) \;>\; a_{j_2} - (x_\mu + r_2 \cdot \delta_\mu) \\ &= a_{j_1} - (x_\mu + r_1 \cdot \delta_\mu)\,. \end{aligned}$$

□

Corollary 2.13. *For K_2 fixed, $\mathcal{O}(M^2)$ triples (x, δ, k) are to be computed, from which the optimal solution can be composed in Stage 2 of the algorithm.*

2.2.4 Stage 2: Optimal Solutions to the Original Problem

From the data computed in Stage 1 of the algorithm, we can now compute an optimal solution to the original problem using a dynamic–programming scheme.

In that, we will compute an optimal solution for each subset $\{1, \ldots, j\}$ of the item set $\{1, \ldots, M\}$ with exactly k intervals for all $k \in \{1, \ldots, K_1\}$. The optimal solution to the original problem instance is among the K_1 optimal solutions computed for $j = M$.

Denote by $opt(i, j, k)$ the value of the optimal solution for the range $\{i, \ldots, j\}$ of items with exactly k intervals. As the result of Stage 1, we have computed the initial values $opt(i, j, 1)$ for all pairs $i < j$. Note that for the trivial case of one single item we have $opt(i, i, 1) = 1$ in the relative case and 0 in the absolute case. For the general case, the following recursion is obvious:

$$\text{opt}(1, j, k) = \min_{1 \le i < j} \{\max\{\text{opt}(1, i, k-1),\ \text{opt}(i+1, j, 1)\}\} \text{ for } k > 1. \tag{2.31}$$

Clearly, we can compute $opt(1, M, k)$ in $\mathcal{O}(M^2)$ time and simultaneously keep track of the corresponding triples which lead to the optimal solution.

Lemma 2.14. *For K_1 and K_2 fixed, Stage 2 of the algorithm requires $\mathcal{O}(M^2)$ asymptotic run time.*

2.2.5 Modification for Error Sums

In order to use the algorithm with the sum of all weighted absolute or relative rounding errors as objective function, some slight modifications will be necessary. In this section, we will picture these changes tersely without being too formal. As the error bounds in Definition 2.8 apply to every single item, this objective function grows linearly with the number of items.

First of all, we have to know, which items are contained in which buckets, and how large is the error sum in each bucket defined in Definition 2.11. As we have saved the biggest items in the buckets $\lambda \in \{0, \ldots, N+1\}$ yet, we know the smallest ones as well, and thus, we can calculate and maintain the sum of errors for each bucket. When δ is increased during the sweep–line method in the inner loop, the error sums in all bins can be easily updated in linear time.

The "2-point-fit"-situation from Lemma 2.9 keeps unaffected. The "equilibrium" situation does not appear using error sums. Thus, only the δ-update $\kappa_\mu(\lambda)$ has to be calculated as denoted by (2.18). The minimum among these values is chosen as increment for δ.
The optimal solutions from Stage 1 for each subinterval $[i; j]$ are stored in a matrix (b_{ij}) as before. Finally, the recursion from Stage 2 must be transformed into

$$\text{opt}(1, j, k) = \min_{1 \le i < j} \Big\{\text{opt}(1, i, k-1) + \text{opt}(i+1, j, 1)\Big\} \text{ for } k > 1. \tag{2.32}$$

2.3 Computational Results

In this Section, we apply Adaptive Rounding to some data distributions that often occur in real world problems. We use the maxima of relative errors as this error measure seems most appropriate for our experiments. In our results, we present the error produced by Adaptive Rounding with parameters K_1 and K_2 to the upper bound shown in Section 2.1.5, to the error value arising from the rounding to $K_1 \cdot K_2$ equidistant rounding values (arithmetic rounding), and the error produced by the rounding to $K_1 \cdot K_2$ arbitrary rounding values. For computation, we use randomly generated numbers as Gaussian, log–normal and uniform distributions on the one hand, and real-world data on the other hand. The latter is given by the different edge lengths from TSP graphs of a couple of TSP instances from TSPLIB (REINELT, 1991).

2.3.1 Gaussian, log–normal and uniform distributions

The Gaussian, log–normal and uniform data distributions have been produced using the GSL (GNU Scientific Library) implementations of two algorithms: the Gaussian distribution was computed using the Ziggurat algorithm (MARSAGLIA & TSANG, 2000) in conjunction with the Mersenne Twister (MT19937) random number generator (MATSUMOTO & NISHIMURA, 1998) with $\sigma = 1.0$. The log–normal and uniform distributions use an implementation of the the RANLUX algorithm (LÜSCHER, 1994), which represents the most reliable source of uncorrelated numbers at that time. More specifically, we have used *ranlux389* which gives the highest level of randomness, with all 24 bits decorrelated. The log–normal distribution was computed with standard parameters $\sigma = 1.0$ and $\zeta = 0.0$ We have applied the above rounding techniques to instances of 500, 1000, and 2000 items distributed with the methods mentioned above in the interval $[100; 100,000]$. Tables 2.1 to 2.3 show the maximum relative errors Δ_{max} using Adaptive Rounding compared to the upper error bounds ub, the errors using equidistant rounding, and those using rounding to K arbitrary values for a set of tuples (K_1, K_2).

2.3.2 Real-world data from TSP instances

Additionally, we applied Adaptive Rounding to real-world data. Therefore, we have extracted pairwise different edge lengths from TSP graphs of a couple of TSP instances. In particular Adaptive Rounding was successfully applied to *LIN318*, *PR1002*, *NRW1379* and *PLA7397* from TSPLIB (REINELT, 1991). *LIN318* appeared 1973 in a paper by Lin&Kernighan, and was first solved in 1980. The 318 point data set arose from a drilling application and contains 643 pairwise different edge lengths. Furthermore, we used the 1002-city problem *PR1002* from Padberg&Rinaldi with 1236 pairwise different edge lengths, the 1379 cities in North Rhine-Westphalia problem *NRW1379* by Bachem&Wottawa having 2284 pairwise different edge lengths, and a programmed logic array problem *PLA7397* by D.S. Johnson with 3175 edge lengths. Table 2.4 shows the maximum relative errors Δ_{max} using Adaptive Rounding compared to the upper error bounds ub, the errors using equidistant rounding, and those using rounding to K arbitrary values for a set of tuples (K_1, K_2) using the instances *LIN318*, *PR1002* and *NRW1379*.

2.3.3 Interpretation of results

As our computational experiments show, Adaptive Rounding is a competitive rounding technique with respect to the resulting rounding error. In particular, it performs well on the randomly generated data profiles that occur in specific real-world applications, and on the real-world data that was extracted from large TSP instances as well. The upper bound on the rounding error given in Section 2.1.5 was never met in any of our experiments. As a matter of fact, Adaptive Rounding does not yield an error as good as rounding to K arbitrary values. However, the resulting errors of Adaptive Rounding and the rounding to K arbitrary values are very close, and might be of no consequence in most cases when using an adequate (K_1, K_2)-combination. As a surplus, Adaptive Rounding provides a special data structure as its output which might be exploited by subsequent procedures as it is for example done in Chapter 4.

Our results show that arithmetic rounding to equidistant values performs quite badly even if the number of rounding values is sufficiently large. For instance, consider the random instances with 500 items from Table 2.1. Here, we need at least 150 rounding values to beat down the rounding error below 2.0. This is nearly a third of the original instance size. With the two other techniques, the largest error yields from the log–normal distribution with $5 \cdot 5 = 25$ rounding values, which is $1/20$ of

		Gaussian				log–normal				uniform			
K_1	K_2	Δ_{max}	ub	equidist.	K val.	Δ_{max}	ub	equidist.	K val.	Δ_{max}	ub	equidist.	K val.
5	5	1.2102	1.5964	35.6399	1.1159	1.3885	1.5964	42.6327	1.2702	1.3006	1.5964	42.5372	1.2164
5	10	1.0797	1.2982	21.0565	1.0433	1.2061	1.2982	21.1092	1.1118	1.1534	1.2982	21.1291	1.0837
5	15	1.0474	1.1988	9.2541	1.0244	1.1369	1.1988	14.3570	1.0630	1.1063	1.1988	14.1071	1.0449
5	20	1.0370	1.1491	9.2541	1.0154	1.0929	1.1491	10.9946	1.0444	1.0814	1.1491	9.6669	1.0299
5	25	1.0295	1.1193	1.4523	1.0109	1.0791	1.1193	8.6183	1.0317	1.0680	1.1193	5.9648	1.0214
10	5	1.0587	1.1991	21.0565	1.0433	1.1385	1.1991	21.1092	1.1118	1.0980	1.1991	21.1291	1.0837
10	10	1.0272	1.0995	9.2541	1.0154	1.0586	1.0995	10.9946	1.0444	1.0501	1.0995	9.6669	1.0299
10	15	1.0156	1.0664	1.4599	1.0080	1.0429	1.0664	7.6338	1.0231	1.0312	1.0664	5.9648	1.0155
10	20	1.0129	1.0498	1.5359	1.0050	1.0312	1.0498	5.8610	1.0140	1.0235	1.0498	5.9648	1.0088
10	25	1.0120	1.0398	1.2340	1.0033	1.0229	1.0398	4.9711	1.0091	1.0213	1.0398	4.6328	1.0058
15	5	1.0479	1.1170	9.2541	1.0244	1.0771	1.1170	14.3570	1.0630	1.0597	1.1170	14.1071	1.0449
15	10	1.0276	1.0585	1.4599	1.0080	1.0279	1.0585	7.6338	1.0231	1.0270	1.0585	5.9648	1.0155
15	15	1.0132	1.0390	1.6936	1.0041	1.0229	1.0390	5.1920	1.0108	1.0174	1.0390	4.6328	1.0070
15	20	1.0064	1.0292	1.2036	1.0021	1.0175	1.0292	4.2564	1.0061	1.0122	1.0292	1.6222	1.0036
15	25	1.0054	1.0234	1.1110	1.0010	1.0135	1.0234	3.5698	1.0031	1.0112	1.0234	1.6236	1.0016
20	5	1.0219	1.0825	9.2541	1.0154	1.0527	1.0825	10.9946	1.0444	1.0400	1.0825	9.6669	1.0299
20	10	1.0224	1.0413	1.5359	1.0050	1.0253	1.0413	5.8610	1.0140	1.0167	1.0413	5.9648	1.0088
20	15	1.0113	1.0275	1.2036	1.0021	1.0158	1.0275	4.2564	1.0061	1.0115	1.0275	1.6222	1.0036
20	20	1.0099	1.0206	1.0863	1.0007	1.0116	1.0206	3.4563	1.0022	1.0079	1.0206	1.7012	1.0012
20	25	1.0099	1.0165	1.0795	1.0000	1.0093	1.0165	2.2861	1.0000	1.0060	1.0165	1.6222	1.0000
25	5	1.0116	1.0637	1.4523	1.0109	1.0394	1.0637	8.6183	1.0317	1.0292	1.0637	5.9648	1.0214
25	10	1.0051	1.0318	1.2340	1.0033	1.0183	1.0318	4.9711	1.0091	1.0143	1.0318	4.6328	1.0058
25	15	1.0113	1.0212	1.1110	1.0010	1.0114	1.0212	3.5698	1.0031	1.0079	1.0212	1.6236	1.0016
25	20	1.0092	1.0159	1.0795	1.0000	1.0083	1.0159	2.2861	1.0000	1.0064	1.0159	1.6222	1.0000
25	25	1.0092	1.0127	1.0614	1.0000	1.0065	1.0127	2.2861	1.0000	1.0046	1.0127	1.6222	1.0000

Table 2.1: Maximum relative error Δ_{max} using Adaptive Rounding compared to the upper error bound ub, the error using equidistant rounding, and a rounding to K arbitrary values. The instances consist of 500 values distributed in $[100; 100,000]$.

		Gaussian				log–normal				uniform			
K_1	K_2	Δ_{max}	ub	equidist.	K val.	Δ_{max}	ub	equidist.	K val.	Δ_{max}	ub	equidist.	K val.
5	5	1.1437	1.5964	3.4471	1.1098	1.4095	1.5964	42.6327	1.2914	1.3409	1.5964	42.5372	1.2393
5	10	1.0692	1.2982	3.4471	1.0445	1.2061	1.2982	21.3427	1.1244	1.1554	1.2982	21.1291	1.0959
5	15	1.0473	1.1988	3.4471	1.0247	1.1466	1.1988	14.3570	1.0735	1.1130	1.1988	14.1071	1.0595
5	20	1.0327	1.1491	3.4471	1.0165	1.1342	1.1491	10.9946	1.0534	1.0908	1.1491	10.0516	1.0376
5	25	1.0279	1.1193	3.4471	1.0120	1.0811	1.1193	9.0372	1.0391	1.0778	1.1193	5.9648	1.0284
10	5	1.0496	1.1991	3.4471	1.0445	1.1385	1.1991	21.3427	1.1244	1.1131	1.1991	21.1291	1.0959
10	10	1.0255	1.0995	3.4471	1.0165	1.0896	1.0995	10.9946	1.0534	1.0553	1.0995	10.0516	1.0376
10	15	1.0149	1.0664	3.4471	1.0090	1.0337	1.0664	7.6712	1.0311	1.0379	1.0664	5.9648	1.0221
10	20	1.0145	1.0498	3.4471	1.0060	1.0352	1.0498	5.9130	1.0205	1.0269	1.0498	5.9648	1.0136
10	25	1.0095	1.0398	3.4471	1.0043	1.0229	1.0398	4.9980	1.0147	1.0220	1.0398	4.6328	1.0099
15	5	1.0317	1.1170	3.4471	1.0247	1.0842	1.1170	14.3570	1.0735	1.0639	1.1170	14.1071	1.0595
15	10	1.0130	1.0585	3.4471	1.0090	1.0414	1.0585	7.6712	1.0311	1.0304	1.0585	5.9648	1.0221
15	15	1.0129	1.0390	3.4471	1.0051	1.0269	1.0390	5.3756	1.0168	1.0216	1.0390	4.6328	1.0117
15	20	1.0068	1.0292	3.4471	1.0032	1.0201	1.0292	4.2564	1.0113	1.0169	1.0292	1.9853	1.0074
15	25	1.0054	1.0234	3.4471	1.0021	1.0155	1.0234	3.5698	1.0082	1.0138	1.0234	1.9853	1.0048
20	5	1.0304	1.0825	3.4471	1.0165	1.0578	1.0825	10.9946	1.0534	1.0509	1.0825	10.0516	1.0376
20	10	1.0150	1.0413	3.4471	1.0060	1.0279	1.0413	5.9130	1.0205	1.0220	1.0413	5.9648	1.0136
20	15	1.0101	1.0275	3.4471	1.0032	1.0184	1.0275	4.2564	1.0113	1.0144	1.0275	1.9853	1.0074
20	20	1.0074	1.0206	3.4471	1.0020	1.0135	1.0206	3.4563	1.0070	1.0118	1.0206	1.9853	1.0044
20	25	1.0037	1.0165	1.1475	1.0011	1.0106	1.0165	2.9226	1.0044	1.0084	1.0165	1.9853	1.0026
25	5	1.0157	1.0637	3.4471	1.0120	1.0459	1.0637	9.0372	1.0391	1.0330	1.0637	5.9648	1.0284
25	10	1.0050	1.0318	3.4471	1.0043	1.0210	1.0318	4.9980	1.0147	1.0166	1.0318	4.6328	1.0099
25	15	1.0047	1.0212	3.4471	1.0021	1.0122	1.0212	3.5698	1.0082	1.0118	1.0212	1.9853	1.0048
25	20	1.0033	1.0159	1.1475	1.0011	1.0104	1.0159	2.9226	1.0044	1.0082	1.0159	1.9853	1.0026
25	25	1.0027	1.0127	1.3245	1.0007	1.0079	1.0127	2.3197	1.0026	1.0059	1.0127	1.9853	1.0015

Table 2.2: Maximum relative error Δ_{max} using Adaptive Rounding compared to the upper error bound ub, the error using equidistant rounding, and a rounding to K arbitrary values. The instances consist of 1000 values distributed in $[100; 100,000]$.

		Gaussian				log–normal				uniform			
K_1	K_2	Δ_{max}	ub	equidist.	K val.	Δ_{max}	ub	equidist.	K val.	Δ_{max}	ub	equidist.	K val.
5	5	1.1731	1.5964	3.6927	1.1228	1.4711	1.5964	42.6505	1.2921	1.3945	1.5964	42.5372	1.2639
5	10	1.0798	1.2982	3.6927	1.0506	1.1851	1.2982	21.3560	1.1231	1.1923	1.2982	21.1291	1.1064
5	15	1.0543	1.1988	3.6927	1.0288	1.1763	1.1988	14.4832	1.0751	1.1234	1.1988	14.1071	1.0642
5	20	1.0384	1.1491	3.6927	1.0199	1.0970	1.1491	11.0780	1.0549	1.0947	1.1491	10.0516	1.0451
5	25	1.0330	1.1193	3.6927	1.0151	1.0858	1.1193	9.0150	1.0403	1.0714	1.1193	8.6834	1.0334
10	5	1.0608	1.1991	3.6927	1.0506	1.1336	1.1991	21.3560	1.1231	1.1304	1.1991	21.1291	1.1064
10	10	1.0336	1.0995	3.6927	1.0199	1.0563	1.0995	11.0780	1.0549	1.0659	1.0995	10.0516	1.0451
10	15	1.0236	1.0664	3.6927	1.0113	1.0491	1.0664	7.6807	1.0332	1.0410	1.0664	6.8241	1.0265
10	20	1.0181	1.0498	3.6927	1.0077	1.0340	1.0498	5.9533	1.0231	1.0311	1.0498	5.9648	1.0181
10	25	1.0112	1.0398	3.6927	1.0056	1.0243	1.0398	4.9700	1.0173	1.0235	1.0398	4.6328	1.0131
15	5	1.0635	1.1170	3.6927	1.0288	1.0893	1.1170	14.4832	1.0751	1.0807	1.1170	14.1071	1.0642
15	10	1.0203	1.0585	3.6927	1.0113	1.0424	1.0585	7.6807	1.0332	1.0370	1.0585	6.8241	1.0265
15	15	1.0146	1.0390	3.6927	1.0065	1.0281	1.0390	5.4257	1.0196	1.0231	1.0390	4.6328	1.0152
15	20	1.0105	1.0292	3.6927	1.0044	1.0207	1.0292	4.2382	1.0135	1.0179	1.0292	3.4467	1.0099
15	25	1.0062	1.0234	3.2887	1.0032	1.0164	1.0234	3.6658	1.0102	1.0146	1.0234	3.4467	1.0074
20	5	1.0257	1.0825	3.6927	1.0199	1.0576	1.0825	11.0780	1.0549	1.0547	1.0825	10.0516	1.0451
20	10	1.0109	1.0413	3.6927	1.0077	1.0297	1.0413	5.9533	1.0231	1.0244	1.0413	5.9648	1.0181
20	15	1.0073	1.0275	3.6927	1.0044	1.0189	1.0275	4.2382	1.0135	1.0165	1.0275	3.4467	1.0099
20	20	1.0044	1.0206	3.2887	1.0029	1.0140	1.0206	3.1328	1.0095	1.0124	1.0206	3.4467	1.0066
20	25	1.0035	1.0165	1.2293	1.0021	1.0115	1.0165	2.9597	1.0068	1.0094	1.0165	2.3461	1.0047
25	5	1.0187	1.0637	3.6927	1.0151	1.0468	1.0637	9.0150	1.0403	1.0419	1.0637	8.6834	1.0334
25	10	1.0082	1.0318	3.6927	1.0056	1.0223	1.0318	4.9700	1.0173	1.0190	1.0318	4.6328	1.0131
25	15	1.0052	1.0212	3.2887	1.0032	1.0120	1.0212	3.6658	1.0102	1.0119	1.0212	3.4467	1.0074
25	20	1.0041	1.0159	1.2293	1.0021	1.0093	1.0159	2.9597	1.0068	1.0087	1.0159	2.3461	1.0047
25	25	1.0029	1.0127	1.4189	1.0014	1.0085	1.0127	2.5060	1.0047	1.0072	1.0127	2.3461	1.0032

Table 2.3: Maximum relative error Δ_{max} using Adaptive Rounding compared to the upper error bound ub, the error using equidistant rounding, and a rounding to K arbitrary values. The instances consist of 2000 values distributed in $[100; 100,000]$.

		TSP LIN318 (643 items)				**TSP PR1002 (1236 items)**				**TSP NRW1379 (2284 items)**			
K_1	K_2	Δ_{max}	ub	equidist.	K val.	Δ_{max}	ub	equidist.	K val.	Δ_{max}	ub	equidist.	K val.
5	5	1.3928	1.8576	173.0000	1.2305	1.3245	2.0669	407.0000	1.2520	1.4392	1.7450	99.0000	1.3176
5	10	1.1813	1.4288	79.0000	1.0913	1.1912	1.5335	207.0000	1.0911	1.2054	1.3725	49.0000	1.1307
5	15	1.1005	1.2859	56.0000	1.0530	1.1119	1.3556	107.0000	1.0497	1.1294	1.2483	32.0000	1.0788
5	20	1.0691	1.2144	32.0000	1.0350	1.0948	1.2667	104.0000	1.0339	1.0893	1.1862	24.0000	1.0553
5	25	1.0492	1.1715	32.0000	1.0256	1.0913	1.2134	57.0000	1.0250	1.0689	1.1490	19.0000	1.0417
10	5	1.1079	1.2599	79.0000	1.0913	1.1331	1.3034	207.0000	1.0911	1.1479	1.2347	49.0000	1.1307
10	10	1.0509	1.1300	32.0000	1.0350	1.0524	1.1517	104.0000	1.0339	1.0674	1.1174	24.0000	1.0553
10	15	1.0373	1.0866	24.0000	1.0193	1.0407	1.1011	57.0000	1.0190	1.0402	1.0782	16.0000	1.0332
10	20	1.0231	1.0650	16.0000	1.0120	1.0334	1.0758	52.0000	1.0122	1.0270	1.0587	12.0000	1.0230
10	25	1.0199	1.0520	16.0000	1.0081	1.0253	1.0607	7.0000	1.0083	1.0196	1.0469	10.0000	1.0172
15	5	1.0665	1.1484	56.0000	1.0530	1.0634	1.1701	107.0000	1.0497	1.0868	1.1356	32.0000	1.0788
15	10	1.0303	1.0742	24.0000	1.0193	1.0315	1.0850	57.0000	1.0190	1.0366	1.0678	16.0000	1.0332
15	15	1.0181	1.0495	16.0000	1.0096	1.0259	1.0567	44.0000	1.0100	1.0217	1.0452	11.0000	1.0198
15	20	1.0125	1.0371	2.0000	1.0059	1.0141	1.0425	7.0000	1.0060	1.0155	1.0339	8.0000	1.0135
15	25	1.0094	1.0297	2.0000	1.0034	1.0123	1.0340	7.0000	1.0040	1.0108	1.0271	7.0000	1.0100
20	5	1.0435	1.1033	32.0000	1.0350	1.0436	1.1173	104.0000	1.0339	1.0592	1.0949	24.0000	1.0553
20	10	1.0189	1.0516	16.0000	1.0120	1.0245	1.0586	52.0000	1.0122	1.0244	1.0474	12.0000	1.0230
20	15	1.0121	1.0344	2.0000	1.0059	1.0183	1.0391	7.0000	1.0060	1.0144	1.0316	8.0000	1.0135
20	20	1.0071	1.0258	2.0000	1.0029	1.0120	1.0293	7.0000	1.0034	1.0097	1.0237	6.0000	1.0092
20	25	1.0061	1.0207	2.0000	1.0009	1.0069	1.0235	7.0000	1.0021	1.0071	1.0190	5.0000	1.0067
25	5	1.0302	1.0791	32.0000	1.0256	1.0337	1.0893	57.0000	1.0250	1.0432	1.0728	19.0000	1.0417
25	10	1.0137	1.0395	16.0000	1.0081	1.0114	1.0447	7.0000	1.0083	1.0182	1.0364	10.0000	1.0172
25	15	1.0074	1.0264	2.0000	1.0034	1.0067	1.0298	7.0000	1.0040	1.0104	1.0243	7.0000	1.0100
25	20	1.0056	1.0198	2.0000	1.0009	1.0061	1.0223	7.0000	1.0021	1.0071	1.0182	5.0000	1.0067
25	25	1.0044	1.0158	2.0000	1.0004	1.0057	1.0179	7.0000	1.0013	1.0053	1.0146	4.0000	1.0049

Table 2.4: Maximum relative error Δ_{max} using Adaptive Rounding compared to the upper error bound ub, the error using equidistant rounding, and a rounding to K arbitrary values using pairwise different edge lengths from 3 TSPLIB instances.

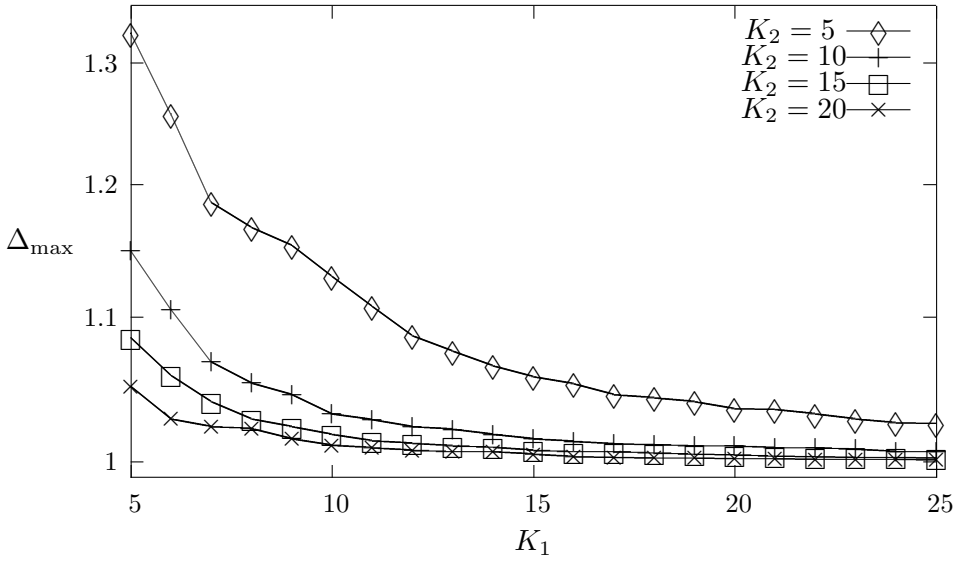


Figure 2.5: Increasing K_1 for $N = 1236$ and fixed $K2$ using pairwise different edge lengths from the 1002-city TSP *PR1002* by Padberg&Rinaldi from TSPLIB (REINELT, 1991). Relative errors were used.

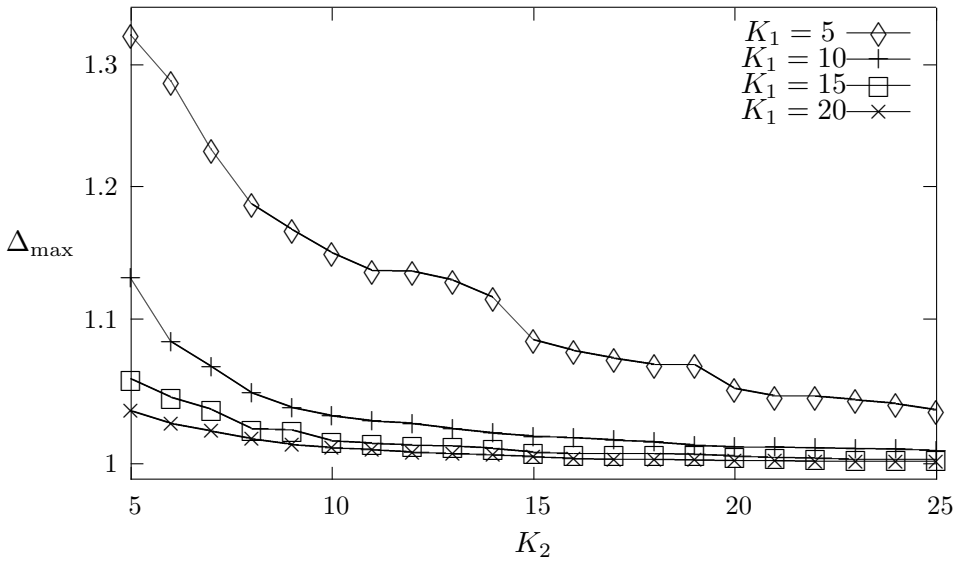


Figure 2.6: Increasing K_2 for $N = 1236$ and fixed $K1$ using edge lengths from the 1002-city TSP *PR1002* by Padberg&Rinaldi from TSPLIB (REINELT, 1991). Relative errors were used.

the original size. Another example for the bad performance of arithmetic rounding are the Gaussian distributions with 1000 or 2000 values respectively. Here, an interesting point is to notice: the largest rounding error is caused by one of the smallest values of the distribution which is rounded down to the first rounding value. Clearly, even with an increase of rounding values, there is no rounding value sweeping below the regarded value. The results is that, even with spending lots of rounding values, the error remains the same.

Furthermore, it should be mentioned that, for each tuple of combinations $((K_1, K_2), (K_2, K_1))$ with $K_1 > K_2$, usually the first one yields a better rounding error. This is due to the fact that the degree of freedom is higher with a large K_1 as the X-values do not require equidistance, whereas the Δ-values,

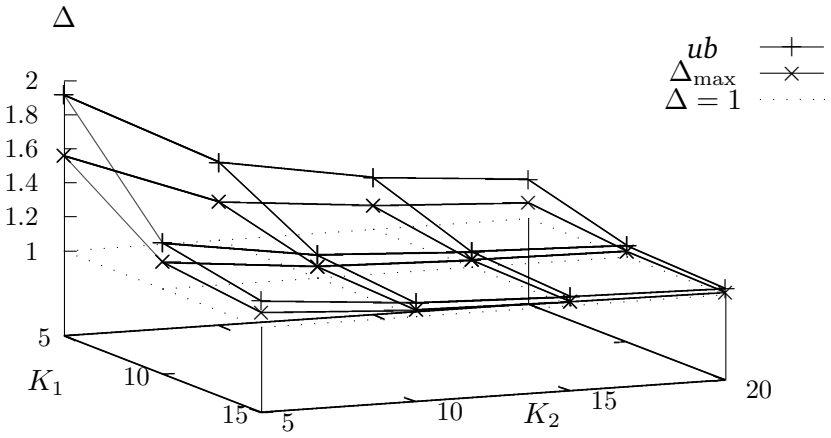


Figure 2.7: Relative error maxima and upper bounds for the instance *PLA7397* from TSPLIB (REINELT, 1991) ($N = 3175$) for several (K_1, K_2)–combinations.

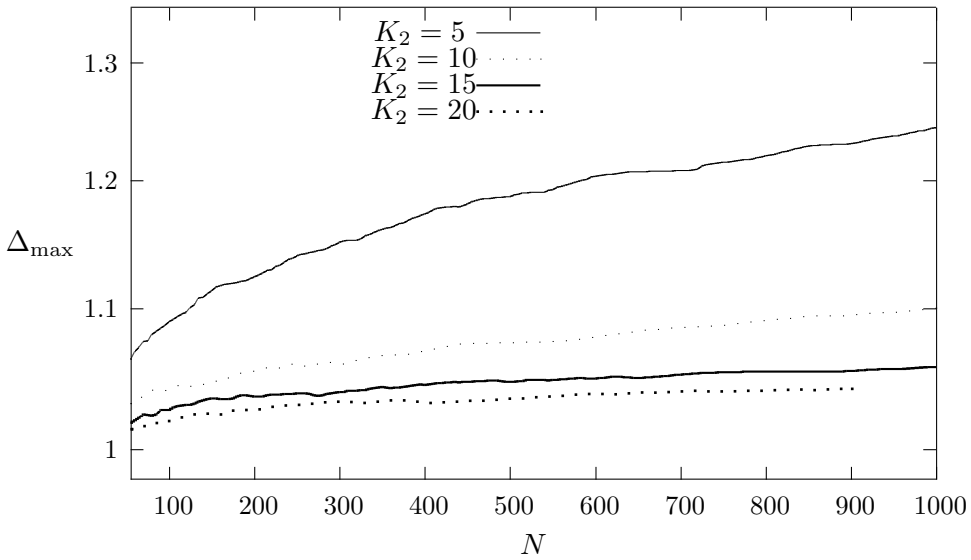


Figure 2.8: Increasing the size N of the original instance for $K_1 = 5$ and fixed K_2, using relative error maxima and uniformly distributed variables within the interval $[100; 100,000]$.

whose number is given by K_2, do. Figures 2.5 and 2.6 show experiments with increasing values for K_1 and K_2, respectively. These were made on the *PR1002* data set from TSPLIB. In Figure 2.5, we regard the relative rounding error for a constant number K_2 of rounding values per interval while increasing the number of intervals K_1. The other way round, in Fig. 2.6, the number K_2 of rounding values is increased while leaving K_1 fixed. In both cases, the error decreases steadily. As one would expect, the error decreases not linearly, but is damped. By means of the diagram in Fig. 2.7, the relative error maximum is illustrated for several (K_1, K_2)–configurations using pairwise different edge lengths from the TSP instance *PLA7397*.

Figure 2.8 shows the error increase subject to rising the number of items N in the original instance. We used uniformly distributed data on the interval $[100; 100,000]$ here, and leave K_1 and K_2 fixed. Expectedly, the error increase is damped like a logarithmic function of N.

2.4 Summary

In this chapter, we introduced several rounding concepts. These may be used for the relaxation of combinatorial optimization problems in order to transform them into an efficiently solvable case. We presented algorithmic techniques for their realization and gave bounds for their maximum rounding errors that are provably tight. We developed a universal algorithm for the Adaptive Rounding Problem, which is reasonably efficient in the worst case if the parameters K_1 and K_2 are chosen relatively small. In our motivating application – the transformation of arbitrary input data into efficiently solvable special cases – these parameters may be freely chosen to obtain a good compromise between run time and the degree of relaxation. Moreover, these parameters are usually chosen small in order to produce sets of classified objects of reasonable cardinality. This is due to the fact that the runtime of subsequent algorithmic techniques is usually polynomial in the cardinality of these classes.

Data profiles from real world applications often show a specific structure like granular data, or Gaussian, uniform, and log–normal distributions. Such typical data profiles were used in our computational results (cf. Section 2.3). With all of these data profiles, the rounding errors are small. Moreover, the rounding procedures perform very well on real-world data that was distilled from several TSP instances. Hence, the degree of the relaxation can be held at a reasonable level in order to avoid negative side effects such as a sudden feasibility in case of originally infeasible problems, or a too large extension of the original solution space.

3 Solution Techniques for High Multiplicity Bin Packing Problems

Damit das Mögliche entsteht,
muß immer wieder das Unmögliche versucht werden.

— Hermann Hesse

In this chapter, we consider the high multiplicity variant of the BIN PACKING PROBLEM (BP), that is the case in which only a fixed number m of different object sizes occur in (high) multiplicities. This is of high practical relevance as – for example – packaging sizes or job lengths in scheduling often satisfy this criterion. The problem is equivalent to the well known one–dimensional CUTTING STOCK PROBLEM (CSP).

Several approaches to solve the problem have been presented in the past. They range from polynomial time approximation schemes (FILIPPI, 2007; GILMORE & GOMORY, 1961, 1963; KARMARKAR & KARP, 1982), mainly based on Linear Programming (LP) relaxations, exact solutions via Branch&Bound (SCHEITHAUER & TERNO, 1995) to heuristic methods (GAU & WÄSCHER, 1996), only to mention a few. MCCORMICK ET AL. (2001) present an exact polynomial time approach for the case $m = 2$. Recently, AGNETIS & FILIPPI (2005) improved and extended it to an approximation algorithm for arbitrary m. The algorithm computes solutions that use at most $m-2$ more bins than the optimum.

Based on this approach, we have further improved the above algorithm for $m = 2$. We present theoretical insights for the exact solution to the m–dimensional HIGH MULTIPLICITY BIN PACKING PROBLEM. Opposed to LP techniques, our algorithmic approach is completely of combinatorial nature. We will study the polyhedral structure of the underlying lattice polytope in order to exploit it in our approach. We will prove bounds on the number of dominating lattice points in the polytope and give some evidence about feasibility of instances. Finally, we present an algorithm for the general HIGH MULTIPLICITY BIN PACKING PROBLEM, that constructs solutions using no more than $OPT + 1$ bins in polynomial time. This represents the best known solution quality delivered by a polynomial approach so far.

We suppose that the reader is familiar with Integer Programming (IP) and polyhedral methods in discrete optimization. For a thorough introduction, we refer to excellent texts on this topic, for example NEMHAUSER & WOLSEY (1988), SCHRIJVER (1987), SCHRIJVER (2003), and WOLSEY (1998), or see ATAMTÜRK (2004) for a short course.

3.1 Preliminaries

3.1.1 The HIGH MULTIPLICITY BIN PACKING PROBLEM

Typically, in the literature the BIN PACKING PROBLEM (BP) is referred to using the objective function to pack all given objects into the minimal number of bins of one fixed size. Besides this definition, we will regard the feasibility variant of the problem: given a set of objects with different sizes (or weights) and multiplicities, is there a feasible assignment of all items to n bins of capacity C each? This gives rise to a more general formulation of the above decision problem.

Definition 3.1 (HIGH MULTIPLICITY BIN PACKING PROBLEM (HMBP))**.** *Let $\boldsymbol{v} \in \mathbb{N}^m$, $\boldsymbol{w} \in \mathbb{R}_+^m$, and $\boldsymbol{C} \in \mathbb{R}_+^m$. A general* HIGH MULTIPLICITY BIN PACKING INSTANCE *$\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{C})$ consists of m classes of items. Each class i contains v_i items of size w_i each. Furthermore, n bins of capacity C_j, $j \in \{1, \ldots, n\}$ are given. The decision variant of the general* HIGH MULTIPLICITY BIN PACKING PROBLEM *consists in determining if there is a distribution of items to the bins, such that no bin capacity is exceeded, and no item is left unpacked. For ease of notation, we will refer to the problem as the* HIGH MULTIPLICITY BIN PACKING PROBLEM *or simply as* (HMBP).

As a special case of (HMBP)*, we denote by $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C, n)$ the case in which all bins have uniform size C. Usually, the goal here is to find a distribution of items to n bins. The optimization variant of the problem is denoted by $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C)$ and asks for a distribution of all items using the least possible number of bins.*

Clearly, the optimization variant of (HMBP) – using the minimization of the number of bins as an objective – may be solved by a sequence of solutions to the decision variant of (HMBP) using for example a binary search technique (see GAREY & JOHNSON (1979) or NEMHAUSER & WOLSEY (1988)).

Without loss of generality, we may assume that all item sizes $w_1, \ldots, w_m$ are pairwise different. Thus, in comparison to BIN PACKING (BP), the input of (HMBP) is somehow compressed. Compared to the input size of (HMBP), the input of (BP) might be exponential in size. Therefore, an algorithm solving (HMBP) in polynomial time would solve (BP) as well in polynomial time, whilst the converse is not necessarily true. In this sense, (HMBP) is harder than (BP). From the fact that (BP) is $\mathcal{NP}$-complete (cf. GAREY & JOHNSON (1979)), it follows immediately that (HMBP) is $\mathcal{NP}$-hard, too. However, to our knowledge it was not known so far, whether (HMBP) admits a solution of size polynomial in the size of its input (MARCOTTE, 1986). In this chapter, we will show that (HMBP) admits a solution of size $\mathcal{O}(m^2)$.

3.1.2 Bin Patterns and Lattice Polytopes

Definition 3.2 (Lattice, Lattice Basis)**.** *A* lattice Λ *in $\mathbb{R}^m$ is an m-dimensional additive free group over $\mathbb{Z}$ which generates $\mathbb{R}^m$ over $\mathbb{R}$. Any linearly independent set of m vectors generating Λ is called a* basis *of the lattice Λ.*

Definition 3.3 (Fundamental Domain, Fundamental Parallelepiped)**.** *Let $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m$ be generators of Λ. The set $\mathcal{D}$ of the form*

$$\mathcal{D} = \left\{ \boldsymbol{x} \in \mathbb{R}^m \middle| \boldsymbol{x} = \beta_1 \boldsymbol{w}_1 + \beta_2 \boldsymbol{w}_2 + \ldots + \beta_m \boldsymbol{w}_m,\ 0 \leq \beta_i < 1, \forall i \in \{1, \ldots, m\} \right\}$$

is called fundamental domain *or* fundamental parallelepiped *for the lattice Λ. $\mathbb{R}^m$ has a canonical Lebesgue measure $L(\cdot)$ assigning the unit volume to the fundamental domain $\mathcal{D}$ for Λ. That is, let $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_m$ be a basis of $\mathbb{R}^m$ with $\mathcal{L} = \mathbb{Z}\boldsymbol{e}_1 \oplus \ldots \oplus \mathbb{Z}\boldsymbol{e}_d$ and $\boldsymbol{x} = x_1\boldsymbol{e}_1 + x_2\boldsymbol{e}_2 + \ldots + x_d\boldsymbol{e}_m$ a point in $\mathbb{R}^m$. Then $L(\boldsymbol{x}) = L(x_1) \cdot L(x_2) \cdot \ldots \cdot L(x_m)$.*

Definition 3.4 (Lattice Polytope)**.** *A convex polytope $\mathcal{P}^\star$ is a* lattice polytope *if there is a lattice Λ such that all vertices of $\mathcal{P}^\star$ are lattice points from Λ.*

Definition 3.5 (Feasible Bin Pattern)**.** *Let $\mathcal{B}$ be an instance of* (HMBP)*. A* bin pattern *$\boldsymbol{s}$ is a vertex $(s_1, \ldots, s_m)^\intercal$ in the canonical integral lattice $\mathbb{Z}^m$ in $\mathbb{R}^m$. $\boldsymbol{s}$ is called* feasible with respect to the instance *$\mathcal{B}$, if $0 \leq s_i \leq v_i$ for all $i \in \{1, \ldots, m\}$. Every bin pattern $\boldsymbol{s}$ induces a weight $w(\boldsymbol{s})$, that is the sum of all items contained therein:*

$$w(\boldsymbol{s}) := w(s_1, \ldots, s_m) := \sum_{i=1}^{m} s_i w_i = \boldsymbol{s}^\intercal \boldsymbol{w}. \tag{3.1}$$

A bin pattern $\boldsymbol{s} := (s_1, \ldots, s_m)^\intercal$ is called feasible with respect to a bin *j, if $w(\boldsymbol{s}) \leq C_j$. A bin pattern that is feasible with respect to $\mathcal{B}$ and to all bins $j \in \{1, \ldots, n\}$ is called* feasible bin pattern*. Consider the simplex*

$$S_j \equiv \left\{ \boldsymbol{x} \in \mathbb{R}^m \middle| \boldsymbol{w}^\intercal \boldsymbol{x} \leq C_j,\ \boldsymbol{x} \geq 0 \right\}, \tag{3.2}$$

the set of bin patterns feasible with respect to bin j is exactly the set of lattice points $S_j \cap \mathbb{Z}^m$ in S_j. These are all integer points contained in the knapsack polytope

$$\mathcal{P}_j^\star \equiv \operatorname{conv}(S_j \cap \mathbb{Z}^m), \tag{3.3}$$

which is itself a lattice polytope by definition.

Note that we chose the canonical lattice $\mathbb{Z}^m$ for Λ above. Alternatively, we might choose the generators of the lattice induced by

$$u_i := \min \left\{ v_i, \max_{\ell \in \mathbb{Z}^\star} \left\{ \ell \mid \ell \cdot w_i \leq C_j \right\} \right\},\ \forall i \in \{1, \ldots, m\}. \tag{3.4}$$

Thus, $\Lambda := \mathbb{Z}/u_1 \times \mathbb{Z}/u_2 \times \ldots \times \mathbb{Z}/u_m$. Then, a lattice point would not reflect the number of items of each type, but the size or weight of items packed of each type. Clearly, there is a bijective transformation between the two lattices. For most of our purposes, it is more convenient to talk in terms of cardinality of sets instead of their size. Therefore, we choose $\Lambda := \mathbb{Z}^m$ in the following if not stated otherwise. However, one should keep in mind the other variant.

Definition 3.6 (Dominating Bin Pattern)**.** *Given two bin patterns $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, $\boldsymbol{s}_2$ is said to* dominate *$\boldsymbol{s}_1$, if $s_{1\ell} \leq s_{2\ell}$, for every $\ell \in \{1, \ldots, m\}$, and $s_{1i} < s_{2i}$ for at least one $i \in \{1, \ldots, m\}$. This immediately implies $w(\boldsymbol{s}_1) \leq w(\boldsymbol{s}_2)$. $\boldsymbol{s}_2$ is called* dominating pattern *for bin j, if it is not dominated by any other bin pattern that is feasible with respect to bin j.*

3.1.3 Integer Programming Formulations

In this section, we give two integer programming formulations of (HMBP) with uniform bin capacities that correspond to classical formulations of the one-dimensional CUTTING STOCK PROBLEM (CSP). The first one originates from KANTOROVICH (1960) and dates from 1939. It uses incidence variables $x_{ij} \in \mathbb{Z}^*$ specifying how many items $0 \leq x_{ij} \leq v_i$ of an item class i are assigned to a bin j. Furthermore, $y_j \in \{0, 1\}$ for $j \in \{1, \ldots, K\}$ is 1, if bin j is used, and 0 otherwise. K is an arbitrary upper bound on the number of bins.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{K} y_j \\
\text{s.t.} \quad & \sum_{i=1}^{m} w_i x_{ij} \leq C \cdot y_j, && \forall j \in \{1,\dots,K\}, \\
& \sum_{j=1}^{K} x_{ij} = v_i, && \forall i \in \{1,\dots,m\}, \\
& x_{ij} \in \mathbb{Z}^*, && \forall i \in \{1,\dots,m\}, \\
& && \forall j \in \{1,\dots,K\}, \\
& y_j \in \{0,1\}, && \forall j \in \{1,\dots,K\}.
\end{aligned}
\tag{3.5}
$$

Every feasible solution, $(x_{1j},\dots,x_{mj})^\intercal$ represents a feasible bin pattern or lattice point x_j in the integral lattice of the knapsack polytope defined by the first condition in the above formulation for any bin $j \in \{1,\dots,K\}$ (see Section 3.1.2). There are several problems occurring with this model: The linear relaxation of (3.5) delivers only the trivial lower bound $v^\intercal w/C$. The upper bound obtained by solving LP relaxations of (3.5) might be quite loose, and the problem contains many symmetric solutions (VANCE ET AL., 1994).

Therefore, a different model has been formulated originally for the CUTTING STOCK PROBLEM (CSP). It is often accredited to GILMORE & GOMORY (1961, 1963), but originally due to EISEMANN (1957). This formulation uses the concept of *bin patterns*, or *cutting patterns* in terms of CSP, explicitly. Instead of assigning an item to a bin, it considers the set of all possible patterns of items a bin could be filled with. Let $\boldsymbol{s}^\ell := (s_1^\ell,\dots,s_m^\ell)^\intercal, \ell \in \{1,\dots,L\}$ denote the ℓ-th feasible bin pattern. Then, the linear program can be formulated as follows:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{\ell=1}^{L} y_\ell \\
\text{s.t.} \quad & \sum_{\ell=1}^{L} s_i^\ell y_\ell = v_i, && \forall i \in \{1,\dots,m\}, \\
& y_\ell \in \mathbb{Z}^*, && \forall \ell \in \{1,\dots,L\}.
\end{aligned}
\tag{3.6}
$$

The objective is to minimize the number of used patterns, and thus implicitly the number of used bins. The first constraint ensures that all items are packed, the second that only an integral number of each pattern s_i^ℓ can be chosen. Note that the number of existing bin patterns might be quite large. The optimal solution value of the linear relaxation of (3.6) is a very good lower bound LB on the optimal number of bins: In a basic optimal solution, there will be at most m non-zero variables, and therefore at most m fractional variables. Rounding all fractional variables up to the next integer results in a packing of a superset of the set of items using less than $LB + m$ bins. Therefore, a solution to the original instance uses at most $LB + m$ bins in the worst case as well.

There is another rounding procedure due to GAU & WÄSCHER (1996) which also satisfies the $LB+m$ bound, but seems to perform much better in practice. It rounds the fractional variables down and handles the remaining items by the *First-Fit Decreasing (FFD)* heuristic. For an experimental survey of this and several more approaches to the Cutting Stock Problem see APPLEGATE ET AL. (2003).

An apparent drawback of the above ILP formulation is the number L of pairwise different bin patterns. This number may become exponential in the number of item sizes. There are several ways to deal with this: GILMORE & GOMORY (1961, 1963), for example, suggested the generation of bin patterns only when needed in order to avoid the explicit listing of all patterns. Here, we will enter another path: The large number of constraints in the above formulation might be drastically reduced

using the set $\tilde{L}$ of *dominating patterns* $\tilde{\boldsymbol{s}}^\ell := (\tilde{s}_1^\ell, \ldots, \tilde{s}_m^\ell)^\intercal$, $\ell \in \{1, \ldots, L\}$ and dropping equality in the first constraint set:

$$\begin{aligned} \text{minimize} \quad & \sum_{\ell=1}^{\tilde{L}} y_\ell \\ \text{s.t.} \quad & \sum_{\ell=1}^{\tilde{L}} \tilde{s}_i^\ell y_\ell \geq v_i, && \forall i \in \{1, \ldots, m\}, \\ & y_\ell \in \mathbb{Z}^*, && \forall \ell \in \{1, \ldots, \tilde{L}\}. \end{aligned} \tag{3.7}$$

Usually, $\tilde{L} \ll L$. Note that equality in the first constraint cannot be requested anymore, but must be replaced by a 'greater equal'. This holds for the following reason: as long as not all bins are completely filled with items from the original instance, i.e. contain dominating patterns, there is always a superset of the original item set packed. As every pattern from the patterns used in formulation (3.6) is dominated by at least one dominating pattern from formulation (3.7), the solution spaces of (3.6) and (3.7) are equivalent modulo dominated patterns. Clearly, every solution to (3.7) can be immediately transformed into a solution to (3.6) by simply omitting

$$r_i := \sum_{\ell=1}^{\tilde{L}} \tilde{s}_i^\ell y_\ell - v_i \tag{3.8}$$

items of type i that have been overpacked. The objective function takes the same value in both cases as formulation (3.7) does not use more bins than (3.6). The linear program (3.7) has many less columns than (3.6), but obviously its linear relaxation yields a lower bound of the same quality. We refer to Lemma 3.13 in Section 3.2.1 for an upper bound on the number of dominating patterns.

3.1.4 Bounds and Feasibility

Lemma 3.7 (Bounds)**.** *Consider an instance $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{C})$ of* (HMBP)*. For each bin $j \in \{1, \ldots, n\}$ of $\mathcal{B}$, there is an upper bound*

$$U_j := \max\left\{\boldsymbol{x}^\intercal \boldsymbol{w} \;\middle|\; \boldsymbol{x} \in \{0, \ldots, v_1\} \times \ldots \times \{0, \ldots, v_m\},\ \boldsymbol{x}^\intercal \boldsymbol{w} \leq C_j\right\}, \tag{3.9}$$

on the load of this bin, and a lower bound

$$L_j := \boldsymbol{v}^\intercal \boldsymbol{w} - \sum_{i=1}^{n} U_i + U_j. \tag{3.10}$$

Proof. U_j is the weight of the maximum possible configuration for bin j. Clearly, no packing of higher weight is admissible for bin j without exceeding the C_j. Any partial sum of U_j's for an arbitrary subset of bins identifies the maximum weight that might be packed into those bins. For a contradiction, suppose $z_j := L_j - \varepsilon, \varepsilon > 0$ to be the weight of the filling of bin j. Packing the maximum amount U_ℓ into all other bins $\ell \in \{1, \ldots, j-1, j+1, \ldots, m\}$, yields

$$\sum_{i=1}^{n} U_i - U_j + z_j = \sum_{i=1}^{n} U_i - U_j + L_j - \varepsilon = \boldsymbol{v}^\intercal \boldsymbol{w} - \varepsilon. \tag{3.11}$$

As we have to place $\boldsymbol{v}^\intercal \boldsymbol{w}$ in total, we have to choose $z_j := L_j$ at least. □

Definition 3.8 (Equivalence of instances)**.** *Consider two* (HMBP) *instances* $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{C})$ *and* $\tilde{\mathcal{B}} = (\boldsymbol{v}, \boldsymbol{w}, \tilde{\boldsymbol{C}})$*.* $\mathcal{B}$ *and* $\tilde{\mathcal{B}}$ *are called* equivalent *if their solution spaces are identical, i.e. if they define the same lattice polytope* $\mathcal{P}^\star$*, that is, their sets of feasible bin patterns are identical.*

Proposition 3.9 (Equivalence of instances)**.** *Consider the* (HMBP) *instances* $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{C})$ *and* $\tilde{\mathcal{B}} = (\boldsymbol{v}, \boldsymbol{w}, \tilde{\boldsymbol{C}})$ *with* $\tilde{\boldsymbol{C}} := \{U_1, \ldots, U_n\}$ *a vector of upper bounds as in Lemma 3.7.* $\mathcal{B}$ *and* $\tilde{\mathcal{B}}$ *are equivalent as their solution spaces of* $\mathcal{B}$ *and* $\tilde{\mathcal{B}}$ *are identical.*

Definition 3.10 (Irreducible instances)**.** *Consider a* (HMBP) *instance* $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{C})$*.* $\mathcal{B}$ *is called* irreducible*, if* $\boldsymbol{C} \equiv \tilde{\boldsymbol{C}}$*, i.e.* $\tilde{\boldsymbol{C}}$ *meets the upper bounds as in Lemma 3.7.*

Figure 3.2 shows such a reduction of $\boldsymbol{C}$ to $\tilde{\boldsymbol{C}}$. In particular, as the solution spaces of $\bar{\mathcal{B}}$ and $\mathcal{B}$ are identical, $\boldsymbol{C}$ might be immediately reduced to $\tilde{\boldsymbol{C}}$ in $\mathcal{O}(m \cdot C_{\max})$. Lemma 3.7 immediately implies a trivial certificate of integer infeasibility as well:

Observation 3.11 (Integer Infeasibility)**.** *Consider a* (HMBP) *instance* $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{C})$*. If*

$$\sum_{i=1}^{n} U_i < \boldsymbol{v}^\intercal \boldsymbol{w}, \tag{3.12}$$

the instance is infeasible.

The following result by McCormick et al. (2001) is about feasibility of (HMBP) instances.

Lemma 3.12 (Feasibility of (HMBP))**.** *Consider a High-Multiplicity Bin-Packing instance* $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C, n)$*.* $\mathcal{B}$ *is feasible if the following conditions are satisfied:*

- *there are integral points* $\boldsymbol{i}_k \in \mathcal{P}, k = 1, \ldots, m+1$*, in the knapsack polytope* $\mathcal{P}$ *corresponding to* $\mathcal{B}$ *such that the simplex* $\mathcal{S}$ *spanned by these points is unimodular,*
- *the point* $\hat{v} = (v_1/n, \ldots, v_m/n)^\intercal$ *belongs to* $\mathcal{S}$*.*

As a partial converse, if $\hat{\boldsymbol{v}} \notin \mathcal{P}$*, then* $\mathcal{B}$ *is infeasible.*

Proof. Define $\mathbf{B}$ to be a $(m+1) \times (m+1)$ matrix with column j being the j-th vertex of $\mathcal{S}$. Each column contains an extra 1 in line $m+1$:

$$\mathbf{B} = \begin{pmatrix} \boldsymbol{i}_1 & \boldsymbol{i}_2 & \ldots & \boldsymbol{i}_{m+1} \\ 1 & 1 & \ldots & 1 \end{pmatrix}.$$

It is well known that the volume of $\mathcal{S}$ spanned by the points $\boldsymbol{i}_k \in \mathcal{P}, k = 1, \ldots, m+1$, equals $(1/m!) \cdot |\det(\boldsymbol{B})|$. Unimodularity of $\mathcal{S}$ is equivalent to $\det(\boldsymbol{B}) = \pm 1$, i.e. $\mathbf{B}$ is unimodular, too. $\hat{v} \in \mathcal{S}$ implies that there are multipliers $\lambda_k \geq 0$ with $\mathbf{B}\lambda = \binom{\hat{\boldsymbol{v}}}{1}$. With $x_k = n \cdot \lambda_k$, this equals

$$\mathbf{B}\boldsymbol{x} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \\ n \end{pmatrix}.$$

Since $\mathbf{B}$ is unimodular, and all v_k and n are integral, Cramer's rule ensures integrality for all x_k as well. Furthermore, $\lambda_k \geq 0$ implies $x_k \geq 0$. We can think of x_k as the quantity of bin pattern k used in a feasible solution $\boldsymbol{x}$ to instance $\mathcal{B}$.

Now, suppose $\mathcal{B}$ is feasible. Let b_k^j denote the number of type k items in bin j. Then, it is

$$b_k^1 + b_k^2 + \ldots + b_k^n = v_k, \quad k = 1, \ldots, m.$$

Summing up and dividing by n shows that $(v_1/n, v_2/n, \ldots, v_m/n) = \hat{\boldsymbol{v}}$ can be written as convex combination of the points $(b_1^j, b_2^j, \ldots, b_m^j)^\intercal, j = 1, \ldots, n$. These points are in $\mathcal{P}$, and so $\hat{\boldsymbol{v}}$ is.

For the partial converse, assume that $\hat{\boldsymbol{v}} \notin \mathcal{P}$. Then $\hat{\boldsymbol{v}}$ cannot be presented as convex combination of vertices from $\mathcal{P}$. At least, one vertex that is not part of $\mathcal{P}$ is needed. This vertex is not a feasible bin pattern, and thus the instance is infeasible. □

For $m = 2$, this is easy to see: the instance is feasible, if and only if $\hat{\boldsymbol{v}} \in \mathrm{conv}(\mathcal{P} \cap \mathbb{Z}^2)$. This is due to the fact that $\mathrm{conv}(\mathcal{P} \cap \mathbb{Z}^2) \neq \emptyset$ may be partitioned into unimodular triangles, of which at least one of them is containing $\hat{\boldsymbol{v}}$. Unfortunately, this does not even hold for $m = 3$ as the following example shows: let $\boldsymbol{v} = (4, 2, 1)^\intercal, \boldsymbol{w} = (6, 10, 15)^\intercal, C = 30$ and $n = 2$. One might easily check that $\hat{\boldsymbol{v}} = (2, 1, 0.5)^\intercal \in \mathcal{P}$, although the instance is still infeasible. That is due to the fact that, in higher dimensions, it is possible to find simplices that do not contain any other integral points than their vertices, but for all of that are not unimodular (cf. HAASE (2000)). For that reason, we will develop a less abstract oracle for the feasibility of (HMBP) in Section 3.4.

3.2 Bin Patterns

Recall the definition of bin patterns from Definition 3.5 and 3.6. The set of lattice points in convex lattice polytopes may be exponential in size, although its cardinality may be computed in polynomial time (BARVINOK & POMMERSHEIM, 1999; GOODMAN & O'ROURKE, 1997). Furthermore, good bounds on the number of lattice points are hard to obtain. As we have seen in the previous section, it is interesting to have a closer look at dominating bin patterns according to Definition 3.6. Therefore, we will develop bounds on the number of bin patterns, and a tight upper bound on the number of dominating patterns in this section.

3.2.1 An upper bound on the number of dominating Bin Patterns

Lemma 3.13 (Upper bound to the number of dominating bin patterns)**.** *Consider an instance $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{C})$ of* (HMBP) *and the associated lattice polytopes $\mathcal{P}_j^\star \equiv \mathrm{conv}(S_j \cap \mathbb{Z}^m)$, where $S_j \equiv \left\{\boldsymbol{x} \in \mathbb{R}^m \middle| \boldsymbol{w}^\intercal \boldsymbol{x} \leq C_j,\ \boldsymbol{x} \geq 0\right\}$. Furthermore, let*

$$u_i := \min\left\{v_i, \max_{\ell \in \mathbb{Z}^\star}\left\{\ell \mid \ell \cdot w_i \leq C_j\right\}\right\},\ \forall i \in \{1, \ldots, m\}, \tag{3.13}$$

that is the maximum number of items of type i that may be exclusively packed into bin j with respect to the total number v_i and the bin's capacity C_j. Furthermore, let $m := dim(\boldsymbol{u})$, $u_{\max} := \max_i\{u_i\}$, and w.l.o.g. $u_{\max} \equiv u_1$. Then,

$$\Omega(\boldsymbol{u}) := \sum_{\sigma=1}^{m}\left\{(-1)^{(\sigma+1)} \sum_{\substack{\sigma \equiv 1 : i_1 = m \\ \sigma > 1 : i_2, \ldots, i_\sigma = 2 \\ i_2 < \ldots < i_\sigma}}^{m} \binom{u_{\max} - \sum_{i_\ell} u_{i_\ell} + m - \sigma}{m-1}\right\} \tag{3.14}$$

is an upper bound on the number of dominating bin patterns for bin j.

Proof. We prove Lemma 3.13 by induction on the dimension. We will illustrate the cases $m = 2$ and $m = 3$ as well as there are some phenomena that occur in dimensions higher than 1 for the first time. Without loss of generality, let $u_{\max} := u_1$ in the following context. Clearly, for $m = 1$, there is exactly one dominating bin pattern, that is

$$\max_{\ell \in \mathbb{Z}^\star} \{\ell \mid \ell \cdot w_1 \leq C_j\}, \text{ and } \binom{u_1}{0} - \binom{1-2}{0} = 1 - 0 = 1 \text{ is correct.}$$

For $m = 2$, there are at most $\min\{u_1 + 1; u_2 + 1\}$ dominating patterns located on or below the straight line between $(u_1, 0)^\intercal$ and $(0, u_2)^\intercal$. We might have at most $u_1 + 1$ different integers in the first component. Regarding the second component, if $u_1 \equiv u_2$, there are at most $u_1 + 1$ different bin patterns in total that vary in both components and are not dominated by other patterns. Finally, if $u_1 > u_2$, this number decreases by $u_1 - u_2$. Clearly, there cannot be more dominating tuples differing in both components. Thus, we obtain

$$\begin{aligned} &\binom{u_1 + 1}{1} - \binom{u_1 - u_2}{1} \\ &= u_1 + 1 - u_1 + u_2 \\ &= u_2 + 1 \\ &= \min\{u_1 + 1; u_2 + 1\} \end{aligned}$$

Now, let $m = 3$. Again, we start with the calculation for $u_1 \equiv u_2 \equiv u_3$: in this case, there are at most $\sum_{i=1}^{u_1+1} i$ dominating bin patterns. For any $u_\ell < u_1$, there are $\sum_{i=1}^{u_1-u_\ell} i$ less possibilities. Thus, we obtain

$$\begin{aligned} &\sum_{i=1}^{u_1+1} i - \sum_{i=1}^{u_1-u_2} i - \sum_{i=1}^{u_1-u_3} i \\ = \; &\frac{(u_1+2)(u_1+1)}{2} - \frac{(u_1-u_2+1)(u_1-u_2)}{2} - \frac{(u_1-u_3+1)(u_1-u_3)}{2} \\ = \; &\binom{u_1+2}{2} - \binom{u_1-u_2+1}{2} - \binom{u_1-u_3+1}{2} \\ = \; &\binom{u_1+2}{2} - \sum_{i=1}^{3} \binom{u_1-u_i+1}{2} \end{aligned}$$

There is the rub: again, let $u_{\max} := u_1$ and $u_2 + u_3 < u_1 - 1$. Then there are dominating patterns that will be cut off twice. We fix this by simply adding the number of those patterns that are subtracted twice. These are exactly

$$\sum_{i=1}^{u_1-u_2-u_3-1} i = \binom{u_1-u_2-u_3}{2} = \binom{u_1-u_2-u_3+m-3}{m-1}.$$

This is due to the following observation: the dominating patterns might be written down as a triangular matrix M of triples like in Figure 3.1 on the right hand side. The cut off patterns also form triangular submatrices starting from the corners of M. Clearly, any overlapping of these is then triangular, too.

Now, let us do the induction step $m \to m+1$. Clearly, if $u_1 \equiv u_2 \equiv \ldots \equiv u_{m+1}$, there are at most

$$\overbrace{\sum_{i_1=1}^{u_1}\sum_{i_2=1}^{i_1}\sum_{i_3=1}^{i_2}\cdots\sum_{i_{m-2}=1}^{i_{m-3}}\sum_{i_{m-1}=1}^{i_{m-2}}}^{m-1} i_{m-1} \;=\; \binom{u_1+m-1}{m-1}. \tag{3.15}$$

W.l.o.g., we assume that $u_{\max} := u_1$. Every $u_\ell < u_1$ reduces (3.15) by

$$\overbrace{\sum_{i_1=1}^{u_1-u_\ell}\sum_{i_2=1}^{i_1}\sum_{i_3=1}^{i_2}\cdots\sum_{i_{m-2}=1}^{i_{m-3}}\sum_{i_{m-1}=1}^{i_{m-2}}}^{m-1} i_{m-1} \;=\; \binom{u_1-u_\ell+m-2}{m-1}. \tag{3.16}$$

Furthermore, we have to guarantee that every pattern is subtracted exactly once. We ensure this by adding the repeatedly subtracted patterns for all pairwise different $(i,j) \in \{1,\ldots,m\} \times \{1,\ldots,m\}$, that overlap, i.e. those with $u_i + u_j < u_1 - 1$. Thus, we have to add a correction term

$$\begin{aligned}
&\overbrace{\sum_{i_1=1}^{u_1-u_i-u_j-1}\sum_{i_2=1}^{i_1}\sum_{i_3=1}^{i_2}\cdots\sum_{i_{m-2}=1}^{i_{m-3}}\sum_{i_{m-1}=1}^{i_{m-2}}}^{m-1} i_{m-1},\ \forall\, i,j \text{ with } u_i+u_j < u_1-1 \\
=\;& \binom{u_1-u_i-u_j+m-3}{m-1}, \qquad\qquad \forall\, i,j \text{ with } u_i+u_j < u_1-1 \\
=\;& \sum_{i=1}^{m}\sum_{j=i+1}^{m}\binom{u_1-u_i-u_j+m-3}{m-1}.
\end{aligned}$$

So, in total, there are

$$\begin{aligned}
&\binom{u_{\max}+m-1}{m-1} - \sum_{i=1}^{m}\binom{u_{\max}-u_i+m-2}{m-1} + \\
&\sum_{i=1}^{m}\sum_{j=i+1}^{m}\binom{u_{\max}-u_i-u_j+m-3}{m-1}
\end{aligned} \tag{3.17}$$

dominating bin patterns, so far. But again, it might have happened that, for some tuples (i,j), the subtracted overlap was added more times than necessary. Thus, we have to consider all triples (i,j,k), and again subtract the excessively added points. After that, we might have subtracted too much, and add again, and so on. This results in an alternatingly signed sum of binomial coefficients of decreasing value.

As there might not be packed more than v_i items i into any bin, and the number of items i cannot exceed $\max\limits_{\ell\in\mathbb{Z}^*}\{\ell \mid \ell \cdot w_i \le C_j\}\ \forall\, i$, if packed exclusively into bin j, we can assume w.l.o.g.

$$u_i := \min\left\{v_i, \max_{\ell\in\mathbb{Z}^\star}\{\ell \mid \ell\cdot w_i \le C_j\}\right\},\ \forall i \in \{1,\ldots,m\}, \tag{3.18}$$

cf. (3.13) Last but not least, it is to show that there is always an injective transformation from the set of dominating bin patterns of an arbitrary instance $\mathcal{B}$ to the set of dominating patterns of the instance

used to count the number of patterns. The above estimation is based upon an integral mesh $\mathcal{M}$ generated on the $(m-1)-$dimensional facet F spanned by the points $(u_1, 0, \ldots, 0)^\intercal, (0, u_2, 0, \ldots, 0)^\intercal, \ldots, (0, 0, \ldots, u_m)^\intercal$. Clearly, this mesh has the highest density that is possible maintaining integrality at the same time. $\mathcal{M}$ is generated by all $m-$tuples between the spanning points, that differ in more than one component and are not dominated, i.e. $(u_1, 0, \ldots, 0)^\intercal$, $(u_1-1, 1, \ldots, 0)^\intercal, (u_1-2, 2, \ldots, 0)^\intercal, \ldots, (1, u_2-1, \ldots, 0)^\intercal, (0, u_2, \ldots, 0)^\intercal$. Then, $(u_1-2, 1, 1, \ldots, 0)^\intercal, (u_1-3, 1, 2, \ldots, 0)^\intercal, (0, 1, u_3-1, \ldots, 0)^\intercal$, and so on. Due to the maximality of facet F, every vertex dominating a vertex of $\mathcal{M}$ would be infeasible with respect to the bin capacity. Vertices that are smaller in at least one component than a vertex from $\mathcal{M}$ are dominated by this (and possibly other vertices from $\mathcal{M}$). Hence, there is always an injective transformation from dominating patterns from an instance $\mathcal{B}$ to the patterns from $\mathcal{M}$. Thus the claim follows. □

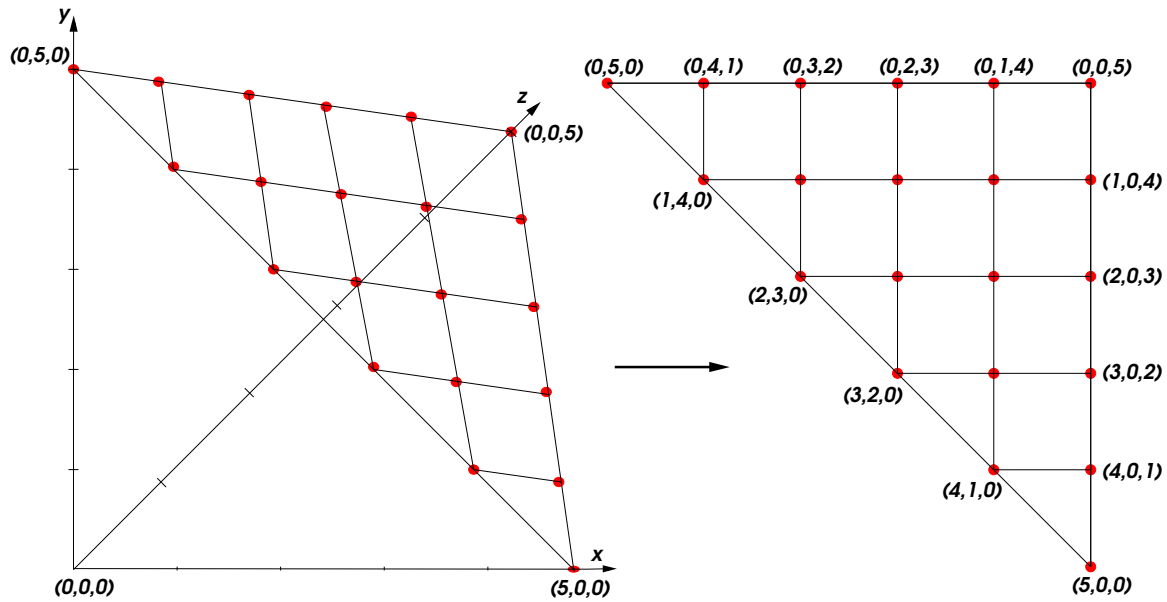


Figure 3.1: Left: 3-dimensional simplex S given by $\{(0,0,0)^\intercal, (5,0,0)^\intercal, (0,5,0)^\intercal, (0,0,5)^\intercal\}$, the dominating integer points are located on the line intersections. Right: resulting 2-dimensional face F, containing the maximal number of dominating integral points of S.

Geometric interpretation

Every bin pattern $(s_1, \ldots, s_m)^\intercal$ may be understood as an integral point in $\mathbb{R}^m$. For illustrative reasons, we regard only one bin of capacity C of a given instance $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C)$. Furthermore, we may assume that $(v_1, 0, \ldots, 0)^\intercal$, $(0, v_2, 0 \ldots, 0)^\intercal$, …, $(0, \ldots, 0, v_m)^\intercal$ are feasible bin patterns with respect to this bin, and $(v_1 + 1, 0, \ldots, 0)^\intercal$, $(0, v_2 + 1, 0 \ldots, 0)^\intercal$, …, $(0, \ldots, 0, v_m + 1)^\intercal$ are no more feasible. The set of all feasible bin patterns corresponds exactly to the intersection points of the integer lattice within the simplex $\mathcal{S}$ defined by the axes and the hyperplane $H : \boldsymbol{v}^\intercal \boldsymbol{w} \leq C$. We count the integral points occurring on the hyperplane H parallel to $(1, 1, \ldots, 1)^\intercal$ during its move within $\mathcal{S}$ starting in the origin $\mathbf{0}$. Clearly, the origin is the first integral point to count. We call this *layer* 0. In layer 1, there are $(1, 0, \ldots, 0)^\intercal, (0, 1, \ldots, 0)^\intercal, (0, 0, \ldots, 1)^\intercal$, that is $\binom{1+m-1}{m-1} = m$ additional integral points. In the second layer, there are $\binom{2+m-1}{m-1}$ integral points, and finally in the ℓ-th layer there are $\binom{\ell+m-1}{m-1}$ integer

points. All these summed up makes $\binom{v_1+m}{m}$ points in total within $\mathcal{S}$. Note that we use the fact here that $\mathcal{S}$ is symmetric, i.e. $v_1 \equiv v_2 \equiv \ldots \equiv v_m$.

As we are only interested in vertices that are not dominated by other vertices, it is sufficient to count the points on H or in direct proximity below H within $\mathcal{S}$ when H has maximum distance from the origin but still contains feasible points. In Figure 3.1, we consider the 3-dimensional simplex $\mathcal{S}$ given by $\operatorname{conv}(\mathbf{0}, (5,0,0)^\intercal, (0,5,0)^\intercal, (0,0,5)^\intercal)$ and the 2-dimensional face $x+y+z=5$, which is an equilateral triangle with side length $\sqrt{50}$ on a hyperplane parallel to $x+y+z=1$. The dominating bin patterns are exactly the integer points on this triangle.

In general, the points $(v_1,0,\ldots,0)^\intercal$, $(0,v_2,0\ldots,0)^\intercal$,...,$(0,\ldots,0,v_m)^\intercal$ together with the origin define an m-dimensional simplex $\mathcal{S}$. The hyperplane H contains the vertices $(v_1,0,\ldots,0)^\intercal, (0,v_2,0,\ldots,0)^\intercal,\ldots,(0,\ldots,0,v_m)^\intercal$. Together with $\mathcal{S}$, it defines an $(m-1)$-dimensional face F spanned by these vertices. Clearly, all dominating points are located on or slightly below F inside $\mathcal{S}$. Points below F which are not dominated by points on F are projected onto F for the purpose of counting all dominating points.

Assuming the maximal number of dominating points, an upper bound on the number of points in $\mathcal{S}$ is $\binom{v_1+m}{m}$. As described above, the number of points in the last layer is at most $\binom{\ell+m-1}{m-1}$. Clearly, H implies maximality: if H is moved further away from $\mathbf{0}$, the gained integer points are not feasible any more. If it is moved in direction to the origin, possibly dominating points are beyond H in $\mathcal{S}$, which are not replaced by a formerly dominated point. This comes from the fact that dominated points might be dominated by more than one point. For example, the origin is dominated by all other points. Therefore, the total number of points would decrease by this operation.

For comparison, we give a bound on the total number of lattice points within $\mathcal{P}^\star$. Note that this bound only holds if we require $\mathcal{P}^\star$ to be simplicial. This is quite close to reality as – for an irreducible instance according to Definition 3.10 and equation (3.13) – the cutting plane given by the capacity constraint cuts the m-dimensional cuboid given by $\boldsymbol{u}_1,\ldots,\boldsymbol{u}_m$ in the axes if $v_\ell^\intercal w_\ell \geq C$,for all $j \in \{1,\ldots,m\}$, i.e. there are more items of each type ℓ that a single bin is able to hold. We are thoroughly aware that the above requirement does not cover the general case. However, general bounds about the number of integer points in polyhedra are much worse (cf. for example COOK ET AL. (1992) or BARVINOK & POMMERSHEIM (1999)).

Corollary 3.14 (Number of lattice points)**.** *Consider an instance $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{C})$ of* (HMBP) *and the associated lattice polytope $\mathcal{P}_j^\star \equiv \operatorname{conv}(S_j \cap \mathbb{Z}^m)$, where $S_j \equiv \left\{\boldsymbol{x} \in \mathbb{R}^m \middle| \boldsymbol{w}^\intercal \boldsymbol{x} \leq C_j,\ \boldsymbol{x} \geq 0\right\}$. Furthermore, let u_i be defined as in (3.13) above, and $u_{\max} := \max_i\{u_i\}$. The total number of lattice points $\left|\mathcal{P}^\star \cap \mathbb{Z}^m\right|$ within $\mathcal{P}^\star$ is bounded by*

$$\sum_{i=0}^{u_{\max}-1} \Omega(\tilde{\boldsymbol{u}}_i) + \sum_{i=1}^{m} u_i, \text{ where } \tilde{u}_{ij} := \begin{cases} u_j - i,\ u_j > i,\ j \in \{1,\ldots,m\}, \\ 0,\ \textit{otherwise}. \end{cases} \tag{3.19}$$

Proof. Lemma 3.13 counts the dominating bin patterns of an instance $\mathcal{B}$. In order to count the total number of lattice points, we move from the layer containing the dominating bin patterns in direction of the origin and estimate the number points in each layer. This can be done by successively decreasing the entries of $\boldsymbol{u}$ down to 0 and summing up the estimates in each layer. We might formula (3.14) for this purpose. Finally, we have to add the non dominating bin patterns located on the axes. These are exactly u_i for each dimension i. □

Formula (3.14) looks far worse than it is in practice. We will give some examples for the number of dominating patterns in Table 3.1. For comparison, we have computed the upper bound the number of total patterns as given in Corollary 3.14, and the number of lattice points in the m–dimensional

m	$\boldsymbol{u} = (u_1, \ldots, u_m)^\intercal$	dom. Ptns.	total Ptns.	$\prod_{i=1}^{m} (u_i + 1)$
2	$(10, 5)^\intercal$	6	36	66
2	$(100, 50)^\intercal$	51	1,476	5,151
3	$(15, 10, 5)^\intercal$	66	234	1,056
3	$(30, 20, 10)^\intercal$	231	1,262	7,161
3	$(60, 40, 20)^\intercal$	861	8,202	52,521
3	$(150, 100, 50)^\intercal$	5,151	113,072	777,801
4	$(20, 15, 10, 5)^\intercal$	856	2,646	22,176
4	$(40, 30, 20, 10)^\intercal$	5,786	29,421	293,601
4	$(80, 60, 40, 20)^\intercal$	42,371	386,721	4,254,201
5	$(25, 20, 15, 10, 5)^\intercal$	11,581	34,725	576,576
5	$(50, 40, 30, 20, 10)^\intercal$	150,161	749,268	14,973,651
5	$(100, 80, 60, 40, 20)^\intercal$	2,151,821	19,321,967	429,674,301
6	$(30, 25, 20, 15, 10, 5)^\intercal$	159,584	473,432	17,873,856
6	$(60, 50, 40, 30, 20, 10)^\intercal$	3,962,167	19,591,679	913,392,711
7	$(35, 30, 25, 20, 15, 10, 5)^\intercal$	2,226,252	6,564,485	643,458,816
7	$(35, 10, 10, 10, 5, 5, 5)^\intercal$	282,680	471,223	10,349,856
7	$(25, 10, 8, 6, 4, 2, 1)^\intercal$	20,432	25,841	540,540
10	$(10, 5, 5, 5, 5, 5, 5, 5, 5, 5)^\intercal$	85,943	150,480	110,854,656
10	$(10, 9, 8, 7, 6, 5, 4, 3, 2, 1)^\intercal$	51,898	70,488	39,916,800

Table 3.1: Upper bounds on the number of dominating bin patterns and the total number of bin patterns according to Lemma 3.13 and Corollary 3.14 using several instances of dimension m with $\boldsymbol{u} = (u_1, \ldots, u_m)^\intercal$ reduced according to equation (3.13). For comparison, the rightmost column gives the number of lattice points in the m–dimensional cuboid spanned by $0, u_1, \ldots, u_m$, that is the bounding box around the lattice polytope arising from the specific instance.

cuboid spanned by $0, u_1, \ldots, u_m$, that is the bounding box around the lattice polytope arising from the specific instance. The more equal the entries of $\boldsymbol{u} = (u_1, \ldots, u_m)^\intercal$ are, the larger is the number of dominating patterns. Usually, the sizes and therefore the numbers of items in (HMBP) instances differ strongly, so the resulting numbers are comparatively small.

3.2.2 Heteroary Codings

For an efficient implementation and a small coding length of bin patterns, we introduce the concept of *heteroary codings*.

Definition 3.15 (Heteroary Coding)**.** *A* heteroary coding $\mathcal{H}^d$ *of dimension* d *is the representation of an interval* $[0; M] \subset \mathbb{N} \cup \{0\}$ *of natural numbers by a number system with a fixed number* d *of digits, where each digit* i *has a different arity* h_i*. We will denote this system by* $\mathcal{H}_{(h_1,\ldots,h_d)}$*.*

$$\mathcal{S}_{(h_1,\ldots,h_d)} := \{0, \ldots, h_1 - 1\} \times \ldots \times \{0, \ldots, h_d - 1\} \tag{3.20}$$

is the resulting set of heteroary numbers *of the coding* $\mathcal{H}_{(h_1,\ldots,h_d)}$*.*

Observation 3.16. *For each heteroary coding $\mathcal{H}_{(h_1,\dots,h_d)}$, we obtain the quantity of representable natural numbers in decimal notation by*

$$|\mathcal{S}_{(h_1,\dots,h_d)}| = \prod_{i=1}^{d} h_i.$$

That is the maximum presentable natural number is $\prod_{i=1}^{d} h_i - 1$. The maximum coding length of $\mathcal{H}_{(h_1,\dots,h_d)}$ is $\sum_{i=1}^{d} \lceil \log_2 h_i \rceil$.

Observation 3.17. *For each heteroary coding $\mathcal{H}_{(h_1,\dots,h_d)}$ and its resulting set of numbers $\mathcal{S}_{(h_1,\dots,h_d)}$ there is a bijective transformation*

$$f : \mathcal{S}_{(h_1,\dots,h_d)} \longrightarrow [0; \prod_{i=1}^{d} h_i - 1] \cap \mathbb{Z}^* \tag{3.21}$$

$$f(\ell_1, \ell_2, \dots, \ell_d) \longmapsto \sum_{i=1}^{d} \left(\ell_i \cdot \prod_{j=i+1}^{d} h_j\right) \tag{3.22}$$

into the decimal (or any arbitrary n-ary) system.

Example. The system $\mathcal{H}_{(7,5,3)}$ generates the set $\mathcal{S} = \{(0,0,0), \dots, (6,4,2)\}$, that is $\{0, \dots, 104\}$ decimal. Counting up from 1 to 10 in $\mathcal{S}$ would look like $(0,0,1), (0,0,2), (0,1,0), (0,1,1), (0,1,2), (0,2,0), (0,2,1), (0,2,2), (0,3,0), (0,3,1)$.

Each instance $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{C})$ of the high multiplicity bin packing problem may be represented by a heteroary coding $\mathcal{H}_{(v_1+1,\dots,v_m+1)}$, where the arities of coding $\mathcal{H}$ correspond exactly to the maximum quantity of items per size minus one. In particular, each bin pattern $\mathcal{P}^\star$ is representable by a heteroary number $\boldsymbol{s} = (s_1, \dots, s_m) \in \mathcal{S}_{(v_1,\dots,v_m)}$ that induces a weight $w(\boldsymbol{s}) = w(s_1, \dots, s_m) := \boldsymbol{s}^\intercal \boldsymbol{w}$ in turn. For each bin $j \in \{1, \dots, n\}$, the set of feasible bin patterns is exactly described in terms of heteroary codings by

$$\mathcal{S}_j := \left\{ \boldsymbol{s} \in \mathcal{S}_{(h_1,\dots,h_d)} \;\middle|\; L_j \le w(\boldsymbol{s}) \le U_j \right\}. \tag{3.23}$$

Definition 3.18 (Componentwise Operations)**.** *Given a heteroary coding $\mathcal{H}_{(h_1,\dots,h_d)}$, the operations $\boldsymbol{s}_1 \oplus \boldsymbol{s}_2$ and $\boldsymbol{s}_1 \ominus \boldsymbol{s}_2$ on $\boldsymbol{s}_1, \boldsymbol{s}_2 \in \mathcal{S}_{(h_1,\dots,h_d)}$ are defined componentwise as in vector arithmetics. In particular, an operation $\boldsymbol{s}_1 \oplus \boldsymbol{s}_2$ resp. $\boldsymbol{s}_1 \ominus \boldsymbol{s}_2$ is called* valid*, if $\boldsymbol{s}_1, \boldsymbol{s}_2$ are chosen such that $\boldsymbol{s}_1 \oplus \boldsymbol{s}_2 \in \mathcal{S}_{(h_1,\dots,h_d)}$ and $\boldsymbol{s}_1 \ominus \boldsymbol{s}_2 \in \mathcal{S}_{(h_1,\dots,h_d)}$.*

3.2.3 Computation of dominating bin patterns

As we have seen from Section 3.2.1, the set of dominating bin patterns is exponential in the size of an (HMBP) instance. Nevertheless, we will give an idea on how the set of dominating bin patterns can be enumerated using the concept of heteroary codings from Section 3.2.2. We define a heteroary coding by $\mathcal{H}_{(u_1,\dots,u_m)}$, where

$$u_i := \min\left\{ v_i, \max_{\ell \in \mathbb{Z}^\star} \left\{ \ell \;\middle|\; \ell \cdot w_i \le C_j \right\} \right\}, \; \forall i \in \{1, \dots, m\}, \tag{3.24}$$

Without loss of generality, we assume the u_i to be sorted in decreasing order by their w_i, so $w_1 > w_2 > \ldots > w_m$. Let $\mathcal{H}_{(u_1,\ldots,u_m)}$ be a heteroary coding sorted as above. Furthermore, for each bin pattern $\boldsymbol{p}_\ell$, we are given the weight function $w(\boldsymbol{p}_\ell) := \boldsymbol{w}^\intercal \boldsymbol{p}_\ell$, and an oracle that decides whether a generated pattern is dominated and can be discarded. This may efficiently done by regarding the residual $\boldsymbol{r} := \boldsymbol{u} \ominus \boldsymbol{p}$ and testing for the rightmost index k of $\boldsymbol{r}$ with $r_k > 0$ whether $w(\boldsymbol{p}_\ell) + w_k < C$. Clearly, if the left hand side is less than C, $\boldsymbol{p}_\ell$ is dominated by $(p_1, \ldots, p_k + 1, \ldots, p_m)$.

We start with codings that contain exactly one component which is maximal according to the above definition of u_i. Clearly, these patterns might be dominated by patterns resulting from an increase of one or more zero components. Creating the dominating patterns from those might be easily achieved by attempting to increase all indexes except i from the left to the right by the maximal possible integral value. We then delete all dominated patterns that have been computed accidentally, and view the resulting set as our start population. This population is put into a queue $\mathcal{Q}$ that is sorted in decreasing lexicographical order by their elements. Starting at the beginning, we successively pick a pattern $\boldsymbol{x}$ from $\mathcal{Q}$ and generate at most $m - 1$ new dominating patterns by the following procedure: For each nonzero component x_i, $i \in \{1, \ldots, m\}$, that has a prefix $(x_1, \ldots, x_i)$ different from its predecessor in $\mathcal{Q}$, we decrease x_i by one and increase another component with index $j > i$ instead by the greatest possible integer such that it satisfies the capacity bound C and the bound u_j. Before the increase of x_j, all components $x_{i+1}, \ldots, x_m$ have to be set to zero in order to avoid duplicates. If the resulting pattern is not dominating yet, increase index $j + 1$ as much as possible, then index $j + 2$, and so on, until no further increase is possible. If the resulting pattern is a dominating pattern, it is stored at the correct location in $\mathcal{Q}$.

As we ensure that the queue does at no time contain dominated patterns, it does not upon termination. Clearly, by applying this approach, every dominating bin pattern is computed exactly once. Algorithm 2 shapes the the above strategy into pseudo code. As a start population, it suffices to use only one single patterns, namely $(u_1 + 1, 0, \ldots, 0)$. This makes things a little bit smarter. As $(u_1 + 1, 0, \ldots, 0)$ is infeasible, we have to delete it in the end.

Algorithm 2 `DominatingPatterns` $(\boldsymbol{v}, \boldsymbol{w}, C)$

initialize: init $\mathcal{H}_{(u_1,\ldots,u_m)}$ and queue $\mathcal{Q} \leftarrow (u_1 + 1, 0, \ldots, 0)$

$\boldsymbol{x} \leftarrow \mathcal{Q}.first()$
repeat
  $k \leftarrow$ leftmost index $\{1, \ldots, m)$ with $x_k \neq z_k$, where $\boldsymbol{z} := \mathcal{Q}.pred(\boldsymbol{x})$
  for $(i = k; i <= m; i++)$
    if $(x_i > 0)$
      $\boldsymbol{y} \leftarrow \boldsymbol{x}$
      $y_i \leftarrow y_i - 1$
      for $(j = i + 1; j <= m; j++)$ $y_j \leftarrow 0$
      for $(j = i + 1; j <= m; j++)$
        $y_j \leftarrow y_j + \min\left\{\lfloor \frac{C - w(\boldsymbol{y})}{w_j} \rfloor, u_j\right\}$
      if $\boldsymbol{y}.isDominating()$ **then** $\mathcal{Q}.save(\boldsymbol{y})$
  $\boldsymbol{x} \leftarrow \mathcal{Q}.next()$
until $\mathcal{Q}.EndOfQueue()$
$\mathcal{Q}.delete(\mathcal{Q}.first())$

3.3 A polynomial algorithm for (HMBP2)

By (HMBP2) we denote the HIGH MULTIPLICITY BIN PACKING PROBLEM with 2 distinct item sizes. An instance is given by a 6-tuple $\mathcal{B} := (v_1, v_2, w_1, w_2, C, n)$. Let $\mathcal{P}^\star := \text{conv}(\mathcal{P} \cap \mathbb{Z}^2)$ be the knapsack lattice polytope for a given instance $\mathcal{B}$ in the following. From Lemma 3.12, it follows for $m = 2$:

Corollary 3.19 (Feasibility of (HMBP2))**.** *For $m = 2$, any* (HMBP) *instance $\mathcal{B} := (\boldsymbol{v}, \boldsymbol{w}, C, n)$ is feasible, if and only if $\bar{\boldsymbol{v}} := (v_1/n, v_2/n)^\intercal \in \mathcal{P}^\star$. If $\mathcal{B}$ is feasible, there is an optimal solution using at most 3 different bin patterns.*

Proof. For every 2-dimensional polytope there exists a (canonical) partition into unimodular triangles. At least one of these triangles contains $\bar{\boldsymbol{v}}$. For the general case, we know from the proof of Lemma 3.12 that – if the (HMBP) instance is feasible – there is an optimal solution using at most $m + 1$ different bin patterns. □

In this case, the number of dominating bin patterns is as small as $\min\{u_1 + 1, u_2 + 1\}$, where $u_i := \min\{v_i, \lfloor C/w_i \rfloor\}$. According to Section 3.1.4, C might be replaced by the upper bound

$$\bar{C} := \max\left\{w^\intercal x \mid \boldsymbol{x} \in \{0, \dots, v_1\} \times \{0, \dots, v_2\},\ w_1x_1 + w_2x_2 \leq C\right\},$$

In the following, we regard the modified (HMBP2) instance $(\boldsymbol{v}, \boldsymbol{w}, \bar{C}, n)$. Let $\bar{n} := \boldsymbol{v}^\intercal \boldsymbol{w}/\bar{C}$ be the optimal solution to this instance dropping integrality. Let $n^\star$ be its optimal integer solution. Obviously, it is $n^\star \geq \lceil \bar{n} \rceil$.

By $\bar{\boldsymbol{v}}$, we denote the point $(v_1/\bar{n}, v_2/\bar{n})^\intercal$ on the straight line $L := \{(x_1, x_2)^\intercal \in \mathbb{R}^2 \ :\ (x_1/v_1, x_2/v_2)^\intercal\}$. By $\tilde{\boldsymbol{v}} := L \cap \partial\mathcal{P}^\star$, we denote the point on L that intersects the boundary $\partial\mathcal{P}^\star$ of $\mathcal{P}^\star$.

Definition 3.20 (Spanning Patterns)**.** *By $\mathcal{G}$ we denote the set of spanning dominating bin patterns of $\mathcal{P}^\star$, that are all patterns that support the convex hull of $\mathcal{P}^\star$.*

Observation 3.21. *The straight line $S := \{x \in \mathbb{R}^2 : w_1x_1 + w_2x_2 = \bar{C}\}$ supports the convex hull of $\mathcal{P}^\star$. $S \cap \mathcal{P}^\star$ defines either an integral vertex, which is a dominating bin pattern from $\mathcal{G}$. Otherwise, $S \cap \mathcal{P}^\star$ defines a facet of $\mathcal{P}^\star$ that is delimited by two dominating bin patterns from $\mathcal{G}$.*

3.3.1 Computation of $\mathcal{G}$

$\mathcal{G}$ may be computed in $\mathcal{O}(\min\{u_1, u_2\})$ sweeping from $\min\{u_1, u_2\}$ to 0: w.l.o.g, for each integral x_1-value starting from $\max\{x_1w_1 \leq C\}$, $x_1 \in \mathbb{Z}^\star$, going down to zero, a maximal x_2-value is computed, such that $x_1w_1 + x_2w_2 \leq C$. Additionally, we compute the gradient $\nabla\overline{\boldsymbol{g}_x\boldsymbol{g}_{x+1}}$ of each line segment

Algorithm 3 `ComputeG`

initialize: $x \leftarrow u_1,\ y \leftarrow \lfloor \frac{\bar{C} - xw_1}{w_2} \rfloor,\ \boldsymbol{g}_{x+1} \leftarrow (x, 0)^\intercal,\ \mathcal{G} \leftarrow \emptyset,\ \delta_\ell \leftarrow -\infty$
while $x > 0$ **do**
  $x \leftarrow x - 1$
  $y \leftarrow \frac{\bar{C} - xw_1}{w_2}$
  $\boldsymbol{g}_x \leftarrow (x, y)^\intercal$
  $\delta_r \leftarrow \delta_l$
  $\delta_\ell \leftarrow \nabla\overline{\boldsymbol{g}_x\boldsymbol{g}_{x+1}}$
  if $\delta_\ell > \delta_r$ **then** $\mathcal{G} \leftarrow (x, y)^\intercal$

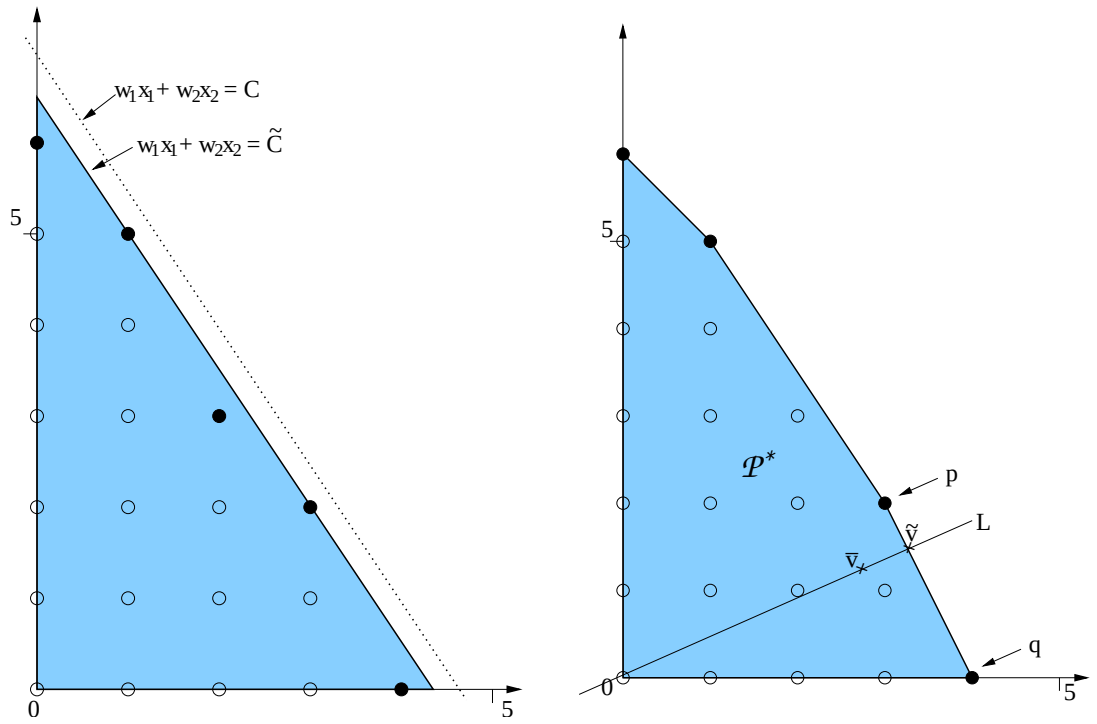


Figure 3.2: The figure on the left hand side shows the set of feasible bin patterns (circles) for the instance $(11, 5, 12, 8, 55, 4)$. The dominating bin patterns are depicted by filled circles. The original bin-capacity $C = 55$ is reduced to $\bar{C} = 52$. On the right hand side the polytope $\mathcal{P}^\star$ is spanned by the set of spanning dominating bin patterns $\mathcal{G}$ (the filled circles) and $\mathbf{0}$. The straight line L and its points $\bar{\boldsymbol{v}} = (2.75, 1.25)^\intercal$ and $\tilde{v} \approx (3.26, 1.48)^\intercal$ are plotted. The two dominating patterns defining the facet of $\mathcal{P}^\star$ containing $\tilde{v}$ are marked p and q.

between neighbored x_1-values. Clearly, we have to add only points $(x_1, x_2)^\intercal$ to $\mathcal{G}$, where the gradient increases from the line segment on the right hand side of $(x_1, x_2)^\intercal$ to that the left hand side. Graphically speaking, we save those points, where the polyline that emerges from connecting all computed points from the right to the left, makes a bend to the left and thus make an active contribution to the convex hull of $\mathcal{P}^\star$.

Clearly, not all vertices of $\mathcal{G}$ have to be computed in order to solve an instance optimally, but only two that define the facet of $\mathcal{P}^\star$ containing $\tilde{v}$.

3.3.2 Exact solution of (HMBP2)

Lemma 3.22. *Given a* (HMBP2) *instance* $(v_1, v_2, w_1, w_2, \bar{C}, n)$. *Further given* $\bar{\boldsymbol{v}}, \tilde{\boldsymbol{v}} \in \mathcal{P}^\star$ *and* $\mathcal{G}$ *as above. Exactly one of the following cases may occur:*

1. $\bar{\boldsymbol{v}} \in \mathbb{Z}^2$: *there is an optimal solution using* n *bin patterns* $\bar{\boldsymbol{v}}$.
2. $\bar{\boldsymbol{v}} \notin \mathbb{Z}^2$: *there are two dominating bin patterns* $\boldsymbol{p}, \boldsymbol{q} \in \mathcal{G}$ *defining the facet of* $\mathcal{P}^\star$ *containing* $\tilde{v}$. *An optimal solution contains multiples of* $\boldsymbol{p}$ *and* $\boldsymbol{q}$ *and at most two more bin patterns* $\boldsymbol{r}_1$ *and* $\boldsymbol{r}_2$. *These are convex combinations from* $\boldsymbol{p}, \boldsymbol{q}$ *and* $\mathbf{0}$.

Proof. See Figures 3.3 and 3.4 for an illustrative support of this proof. We will refer to them where applicable.

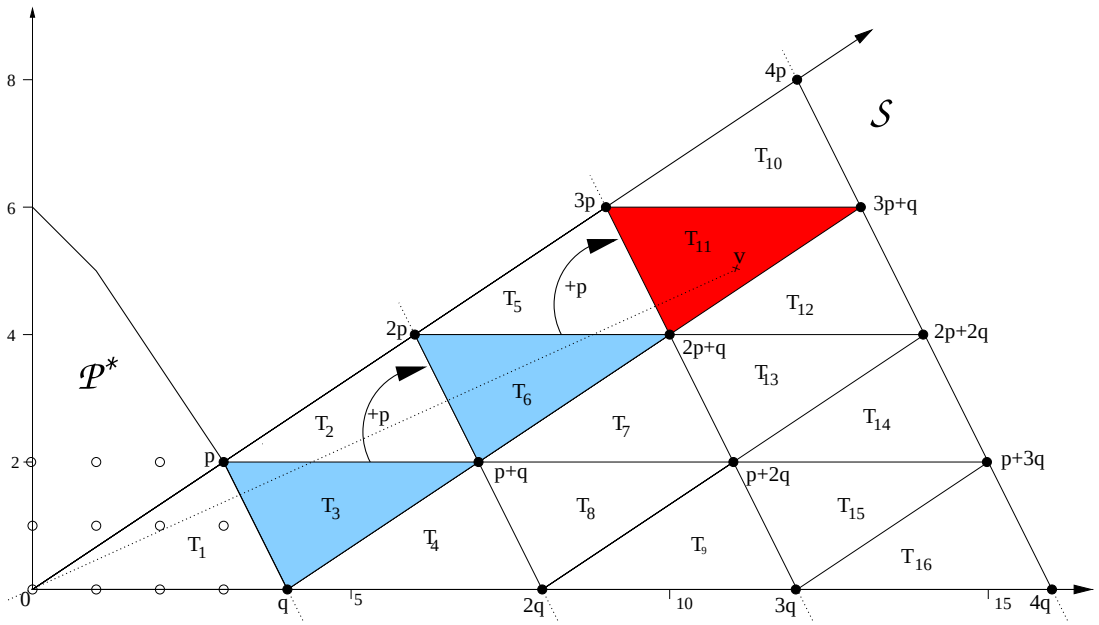


Figure 3.3: Triangulation of the pointed cone $\mathcal{S}$ into α- and β-triangles $T_1 = (\boldsymbol{0}, \boldsymbol{p}, \boldsymbol{q}), T_2 = (\boldsymbol{p}, 2\boldsymbol{p}, \boldsymbol{p}+\boldsymbol{q}), T_3 = (\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{p}+\boldsymbol{q}), \ldots$.

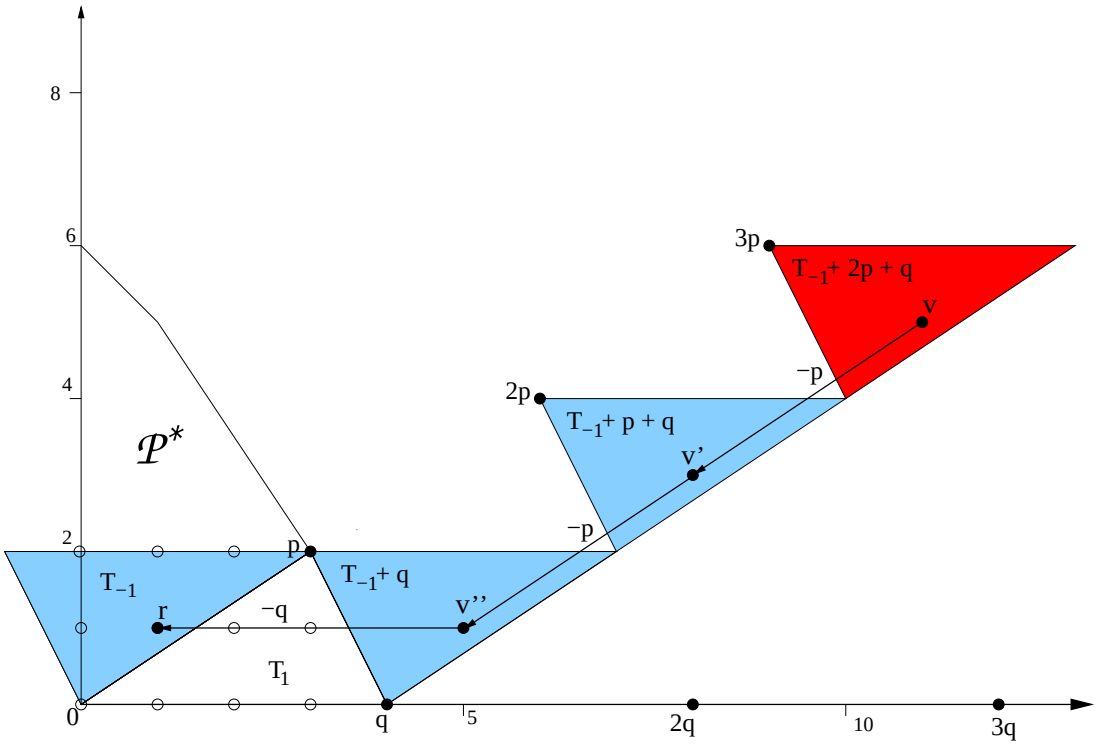


Figure 3.4: Construction of a feasible solution of HMBP2: the triangle T_ℓ containing the vertex $\boldsymbol{v}$ is translated by a combination of $-\boldsymbol{p}$ and $-\boldsymbol{q}$ moves to either the triangle T_1 (if $\boldsymbol{v}$ is located in an α-triangle), or to the triangle T_{-1} (if $\boldsymbol{v}$ is located in a β-triangle). The translation is illustrated above by the sequence $\boldsymbol{v} \to \boldsymbol{v}' \to \boldsymbol{v}'' \to \boldsymbol{r}$. Clearly, all nonnegative residuals $\boldsymbol{r}$ are feasible bin patterns. If $\boldsymbol{r}$ contains negative entries, we project them to zero in order to obtain a feasible pattern. The negative components are subtracted from the patterns $\boldsymbol{p}$ or $\boldsymbol{q}$, respectively.

As $\bar{\boldsymbol{v}} \in \mathcal{P}^\star$, the instance is feasible. If $\bar{\boldsymbol{v}} \in \mathbb{Z}^2$, things are easy, because $\bar{\boldsymbol{v}}$ itself is a feasible but not necessarily dominating bin pattern. A feasible solution with n bins may be built by filling each bin with pattern $\bar{\boldsymbol{v}}$.

For the case $\bar{\boldsymbol{v}} \notin \mathbb{Z}^2$, consider the simplex $\mathcal{S}$ defined by $\mathbf{0}, \boldsymbol{p}$ and $\boldsymbol{q}$. Clearly, $\bar{\boldsymbol{v}} \in \mathcal{S}$. We propose a triangulation of $\mathcal{S}$ as follows: triangle T_1 is given by $\mathcal{P}^\star \cap \mathcal{S}$. All other triangles are generated as follows: The straight line spanning $\mathcal{S}$ and containing $\boldsymbol{p}$ is translated to positions $\boldsymbol{q}, 2\boldsymbol{q}, 3\boldsymbol{q}, 4\boldsymbol{q}$, and so on. The straight line spanning $\mathcal{S}$ and containing $\boldsymbol{q}$ is translated to positions $\boldsymbol{p}, 2\boldsymbol{p}, 3\boldsymbol{p}, 4\boldsymbol{p}$, and so on. Finally, the straight line induced by $\overline{\boldsymbol{pq}}$ is translated to positions $2\boldsymbol{q}, 3\boldsymbol{q}, 4\boldsymbol{q}, \ldots$. Figure 3.3 shows such a triangulation considering the concrete example from Figure 3.2. Clearly, all triangles in $\mathcal{S}$ arose from T_1 by $\boldsymbol{p}$- and $\boldsymbol{q}$-moves and by mirroring T_1 on the axis induced by $\overline{\boldsymbol{pq}}$. We call triangles with the same orientation as T_1 α-*triangles*, whereas mirror-inverted triangles are called β-*triangles*.

If $\boldsymbol{v}$ is located in an α-triangle, a solution to the instance may be easily generated as follows: regard the lower left corner $\boldsymbol{t}_\ell$ of the triangle T_ℓ, which contains $\boldsymbol{v}$. The corner $\boldsymbol{t}_\ell$ has the coordinates $\lambda \cdot \boldsymbol{p} + \mu \cdot \boldsymbol{q},\ \lambda, \mu \in \mathbb{Z}^\star$. The residual $\boldsymbol{r} := \boldsymbol{v} - \lambda \cdot \boldsymbol{p} - \mu \cdot \boldsymbol{q}$ located in T_1 is integral and thus is a feasible bin pattern as well. Therefore, a feasible solution contains λ dominating patterns $\boldsymbol{p}, \mu$ dominating patterns $\boldsymbol{q}$ and one dominated pattern $\boldsymbol{r}$ that is contained in T_1. Clearly, $\boldsymbol{r}$ is a convex combination from $\mathbf{0}, \boldsymbol{p}$ and $\boldsymbol{q}$. As $\boldsymbol{p}$ and $\boldsymbol{q}$ are maximal with respect to $\mathcal{S}$, there cannot be any solution using less bins.

If $\boldsymbol{v}$ is contained in a β-triangle T_ℓ, we regard the number of $\boldsymbol{p}$- and $\boldsymbol{q}$-moves that translate T_{-1} onto T_ℓ. Again, any feasible solution contains λ patterns $\boldsymbol{p}$ and μ patterns $\boldsymbol{q}$ that are necessary to translate T_{-1} onto T_ℓ. The other way round, $\boldsymbol{v}$ is translated onto a point $\boldsymbol{r}$ located in T_{-1} using the reverse move $-\lambda \cdot \boldsymbol{p} - \mu \cdot \boldsymbol{q}$.

If the residual $\boldsymbol{r} \in T_{-1}$ has all nonnegative components, we may simply use $\boldsymbol{r}$ as an additional bin pattern in the solution. This is due to the fact, that all nonnegative integral points of T_{-1} are dominated by points of T_1 and thus are feasible bin patterns. If $\boldsymbol{r} \in T_{-1}$ contains negative entries, we project these to zero and obtain $\boldsymbol{r}'$. Clearly, the projected pattern $\boldsymbol{r}'$ is feasible as it is located in T_1. We choose $\boldsymbol{r}'$ in the solution. The projected, originally negative components $\boldsymbol{r}' - \boldsymbol{r}$ are subtracted from patterns $\boldsymbol{p}$ and $\boldsymbol{q}$, where applicable. Clearly, these entries may be subtracted either from a $\boldsymbol{p}$ or a $\boldsymbol{q}$ pattern as $T_1 \cup T_{-1}$ span a parallelogram, and therefore the absolute value of the negative component cannot be greater than the corresponding component of either $\boldsymbol{p}$ or $\boldsymbol{q}$.

Finally, we have to show that the constructed solution is optimal for the original instance $(v_1, v_2, w_1, w_2, \bar{C}, n)$: All patterns used in the above construction are in $\mathcal{P}^\star$, and thus are feasible for the original instance. As $\boldsymbol{p}$ and $\boldsymbol{q} \in \partial\mathcal{P}^\star$ and $\mathcal{P}^\star$ is convex, there cannot be any dominating patterns $\tilde{\boldsymbol{p}}$ and $\tilde{\boldsymbol{q}} \in \mathcal{P}^\star \setminus \{\boldsymbol{p}, \boldsymbol{q}\}$ spanning at least $\mathcal{P}^\star \cap \mathcal{S}$. As $\boldsymbol{p}$ and $\boldsymbol{q}$ are maximal with regard to $\mathcal{P}^\star$, there cannot be any solution using less patterns than the one constructed above. □

Algorithm 4 computes an optimal solution to (HMBP2) straightforwardly. Figure 3.2 depicts the vertices $\tilde{\boldsymbol{v}}$ and $\bar{\boldsymbol{v}}$ in the characteristic polytope $\mathcal{P}^\star$ and shows the determination of the facet defining points $\boldsymbol{p}$ and $\boldsymbol{q}$. Figure 3.3 shows the resulting triangulation of the cone $\mathcal{S}$. In Figure 3.4 the construction of a feasible solution according to Algorithm 4.

Note that the problem to compute λ, μ, and $\boldsymbol{r}$ amounts to computing a solution to a Diophantine equation, that is an equation with an all integral solution, of the form

$$\lambda \boldsymbol{p} + \mu \boldsymbol{q} = \boldsymbol{v} - \boldsymbol{r}, \text{ where } \lambda, \mu \in \mathbb{Z}_+. \tag{3.25}$$

In case of only 2 distinct bin patterns, this can be easily achieved by computing a fractional solution to the above equation without the last subtrahend $\boldsymbol{r}$. Basically, we take the integral parts of λ and μ, and compute the residual $\boldsymbol{r}$ as the sum of the fractional parts of λ and μ, which is clearly integral.

Algorithm 4 `ExactHMBP2`

determine $\boldsymbol{p}, \boldsymbol{q} \in \mathcal{P}^\star$ spanning a triangle $T = \text{conv}(\mathbf{0}, \boldsymbol{p}, \boldsymbol{q})$ containing $\bar{\boldsymbol{v}}$
if there are no such patterns **break** ($\rightarrow$ infeasibility detected)
solve $\boldsymbol{v} = \bar{\lambda}\boldsymbol{p} + \bar{\mu}\boldsymbol{q}$, with $\bar{\lambda}, \bar{\mu} \in \mathbb{Q}^+$
choose $\lambda := \lfloor \bar{\lambda} \rfloor$ patterns $\boldsymbol{p}$ and $\mu := \lfloor \bar{\mu} \rfloor$ patterns $\boldsymbol{q}$ in an optimal solution
$\boldsymbol{r} := (\bar{\lambda} - \lambda)\boldsymbol{p} + (\bar{\mu} - \mu)\boldsymbol{q}$
if $\boldsymbol{r} \equiv 0$ **then break**
if $\boldsymbol{r} \in T_1$ **then** add pattern $\boldsymbol{r}$ to the optimal solution
else add $\boldsymbol{r}_1, \boldsymbol{r}_2 \in dom(T)$ such that $\boldsymbol{r} = \boldsymbol{r}_1 + \boldsymbol{r}_2$

Lemma 3.23. *An optimal solution to* (HMBP2) *can be computed in* $\mathcal{O}(log\ u)$*, where* $u := \min\{u_1 + 1, u_2 + 1\}$ *and* $u_i := \min\{v_i, \lfloor C/w_i \rfloor\}$.

Proof. The determination of $\boldsymbol{p}$ and $\boldsymbol{q}$ is done in $\mathcal{O}(log\ u)$. The fractional solutions $\bar{\lambda}$ and $\bar{\mu}$ and the residual $\boldsymbol{r}$ are computed in $\mathcal{O}(1)$. The computation of the patterns $\boldsymbol{r}_1, \boldsymbol{r}_2$ can be done in $\mathcal{O}(log\ u)$. □

Note that the optimization variant of the above problem may be optimally solved using Algorithm 4 as well. For this purpose, it can be modified as follows: in the first step $\bar{\boldsymbol{v}}$ is replaced by $\tilde{\boldsymbol{v}}$, that is the intersection point of the line connecting $\boldsymbol{v}$ and the origin and $\mathcal{P}^\star$ (see Figure 3.2). As $\bar{\boldsymbol{v}}$ must be in $\mathcal{P}^\star$ to preserve feasibility, we have to choose n accordingly. Clearly, the choice of $n := \lfloor \boldsymbol{v}/\bar{\boldsymbol{v}} \rfloor$ provides the minimum n that is contained in $\mathcal{P}^\star$. The rest is straightforward.

3.4 Solution of general (HMBP)

Based on the 2–dimensional case, we will develop a modus operandi for the general high multiplicity case. First, we will present theoretical aspects about feasibility and solution of (HMBP) in higher dimensions. Based on these considerations, we will develop an efficient technique to generate solutions to any instance $\mathcal{B}$ of (HMBP) using no more than $n + 1$ bins, even if n is the minimum number of bins. Regarding the optimization variant of (HMBP) our approach generates solutions using no more than $OPT + 1$ bins, where OPT is the optimal solution value. This represents the best known provable solution quality delivered by an efficient approach so far. We will show that – under certain circumstances and with high probability – optimal solutions are attained.

3.4.1 Characteristic Polytopes and Feasibility

Given an arbitrary (HMBP) instance $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C, n)$, our goal is to show that $\mathcal{B}$ is feasible if and only if $\boldsymbol{v}/n \in \mathcal{P}^\star$, where $\mathcal{P}^\star$ is a lattice polytope implied by $\mathcal{B}$. But what does "implied by $\mathcal{B}$" mean more precisely?

Once more, recall the example from the end of Section 3.1.4: Given an instance $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C, n)$ with $\boldsymbol{v} = (4, 2, 1)^\intercal$, $\boldsymbol{w} = (6, 10, 15)^\intercal$, $C = 30$ and $n = 2$. We consider the the lattice polytope $\mathcal{P}^\star$ defined by the simplex $\mathcal{S} := \text{conv}(\mathbf{0}, (5, 0, 0)^\intercal, (0, 3, 0)^\intercal, (0, 0, 2)^\intercal)$ consisting of the maximum number of items that are packageable without exceeding the capacity limit in each dimension. At the same time, $\mathcal{S}$ includes all feasible bin patterns with respect to $\mathcal{B}$. Clearly, $\tilde{\boldsymbol{v}} = \boldsymbol{v}/2 = (2, 1, 0.5)^\intercal \in \mathcal{S}$, but there is no solution with only 2 bins. Figure 3.5 shows the simplex $\mathcal{S}$ which contains the point $\tilde{\boldsymbol{v}}$.

So, what does happen here? The problem is that there is no possible configuration of items at all that fully exploits the given capacity of 30. Although $(5, 0, 0)^\intercal$, $(0, 3, 0)^\intercal$ and $(0, 0, 2)^\intercal$ are feasible bin patterns which attain the capacity of 30 each, they cannot be used due to the shortage of items

available. So it does not make sense at all to check, if $\hat{\boldsymbol{v}}$ is located in $\mathcal{P}^\star$, which is solely defined by the convex hull over all feasible patterns. In our example, the maximal weight of a potential configuration is 28 which is induced by the pattern $(3,1,0)^\intercal$. So, we may reduce the size of C to 28 without losing potentially feasible patterns. This makes $\mathcal{B}$ irreducible according to Definition 3.10. With $C = 28$, the lattice polytope $\mathcal{P}^\star$ shrinks automatically to $\mathrm{conv}(\mathbf{0}, (4,0,0)^\intercal, (0,2,0)^\intercal, (0,0,1)^\intercal, (3,1,0)^\intercal, (2,0,1)^\intercal, (1,2,0)^\intercal)$. This immediately results in $\hat{\boldsymbol{v}}$ being not part of $\mathcal{P}^\star$ anymore. In this case, the feasibility oracle's answer would be suitable. This is depicted in Figure 3.6.
The other way round: if the above instance has a solution with 3 bins, we might choose patterns $(4,0,0)^\intercal$, $(0,2,0)^\intercal$ and $(0,0,1)^\intercal$ in a solution. Clearly, these are dominated by the feasible patterns $(5,0,0)^\intercal$, $(0,3,0)^\intercal$, and $(0,0,2)^\intercal$ that would solve the above instance with $\boldsymbol{v} = (5,3,2)^\intercal$ and $n = 3$. Regarding the case $n = 2$ with $\mathcal{P}^\star$ being the simplex spanned by $\mathbf{0}$, $(5,0,0)^\intercal$, $(0,3,0)^\intercal$ and $(0,0,2)^\intercal$, the feasibility oracle would deliver "true" for the case $\boldsymbol{v} = (4,2,1)^\intercal$, and "false" for the case $\boldsymbol{v} = (5,3,2)^\intercal$ although both instances are infeasible.

Concludingly, the claim for irreducible instances is essential to obtain reliable statements. As the capacity bound might be met in one dimension, but not in others, it is generally not sufficient to claim irreducibility. In fact, there is the requirement to shrink $\mathcal{P}^\star$ by reducing the vertices of $\mathcal{P}^\star$ in each dimension to the maximally taken values

$$u_i := \min\{v_i, \lfloor \frac{C}{w_i} \rfloor\}. \tag{3.26}$$

Potentially, this implies a shrinkage of C yet.

Definition 3.24 (Characteristic Polytope)**.** *Let $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C, n)$ be an irreducible* (HMBP) *instance. Let $\mathcal{P} \equiv \left\{\boldsymbol{x} \in \mathbb{R}^m \;\middle|\; \boldsymbol{w}^\intercal \boldsymbol{x} \leq C,\ 0 \leq x_i \leq v_i\ \forall i \in \{1,\dots m\}\right\}$. Then $\mathcal{P}^\star := \mathrm{conv}(\mathcal{P} \cap \mathbb{Z}^m)$ is called the* characteristic polytope *of $\mathcal{B}$. In particular, $\mathcal{P}^\star$ is a lattice polytope containing all feasible bin patterns that might be effectively taken in a solution.*

Lemma 3.25 (Feasibility of (HMBP))**.** *Consider an* (HMBP) *instance $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C, n)$. Let $\mathcal{P}^\star$ be its characteristic polytope. $\mathcal{B}$ is infeasible if $\boldsymbol{v}/n \notin \mathcal{P}^\star$. Otherwise, $\mathcal{B}$ might be feasible.*

Proof. If $\mathcal{B}$ is infeasible, $\boldsymbol{v}$ cannot be assembled from a combination of exactly n feasible bin patterns. Therefore, $\boldsymbol{v}/n$ is not a convex combination of patterns in $\mathcal{P}^\star$, and thus is outside $\mathcal{P}^\star$.
If $\boldsymbol{v}/n \notin \mathcal{P}^\star$, there is no convex combination of n vertices from $\mathcal{P}^\star$ that generates $\boldsymbol{v}/n$. Therefore at least one vertex outside of $\mathcal{P}^\star$ is necessary to assemble $\boldsymbol{v}$ from n vertices. Therefore, $\mathcal{B}$ is infeasible. □

For several reasons, we cannot make the converse statement about feasibility here. As things are more complex in the case $\boldsymbol{v}/n \in \mathcal{P}^\star$, we are going to illuminate feasibility issues in the forthcoming section.

3.4.2 Near optimal solution of general (HMBP) instances

In this section, we will show that – under certain circumstances – an optimal solution to a feasible (HMBP) instance $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C, n)$ can be composed of at most $m+1$ lattice points from the characteristic polytope $\mathcal{P}^\star \in \mathbb{R}^m$. In either case, we are able to construct a solution using $n+1$ bins and $\mathcal{O}(m)$ lattice points from $\mathcal{P}^\star \in \mathbb{R}^m$. As shown in Section 3.1.2, the number of dominating lattice points in $\mathcal{P}^\star$ might be quite large. In order to narrow the set of lattice points that are needed to compose an optimal solution to a given instance $\mathcal{B}$ to a minimum, we regard only lattice points from that facet $\mathcal{F}$ of $\mathcal{P}^\star$ which is intersected by the straight line defined by $\mathbf{0}$ and $\boldsymbol{v}$. This makes sense because $\mathcal{P}^\star$ is convex and the dominating patterns defining $\mathcal{F}$ are maximal with respect to the vector $\boldsymbol{v}$.

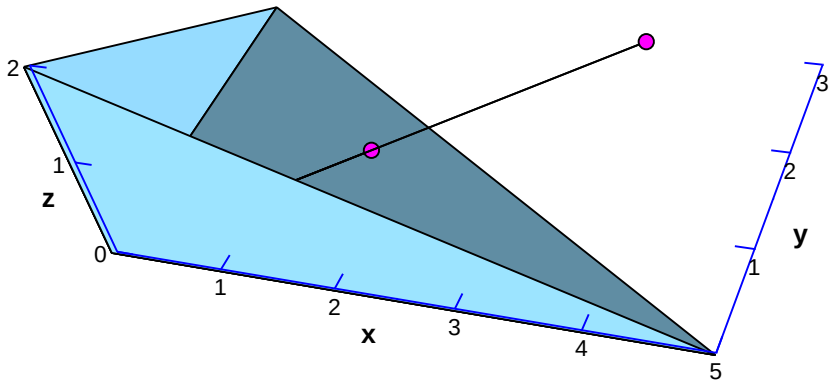


Figure 3.5: We consider an instance $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C, n)$ with $\boldsymbol{v} = (4, 2, 1)^\intercal$, $\boldsymbol{w} = (6, 10, 15)^\intercal$, $C = 30$ and $n = 2$. If the lattice polytope $\mathcal{P}^\star$ consisting of all feasible bin patterns equals the simplex $\mathcal{S} := \operatorname{conv}(0, (5, 0, 0)^\intercal, (0, 3, 0)^\intercal, (0, 0, 2)^\intercal)$, the point $\boldsymbol{v}/2 = (2, 1, 0.5)^\intercal$ is contained in $\mathcal{S}$, although $\mathcal{B}$ is infeasible.

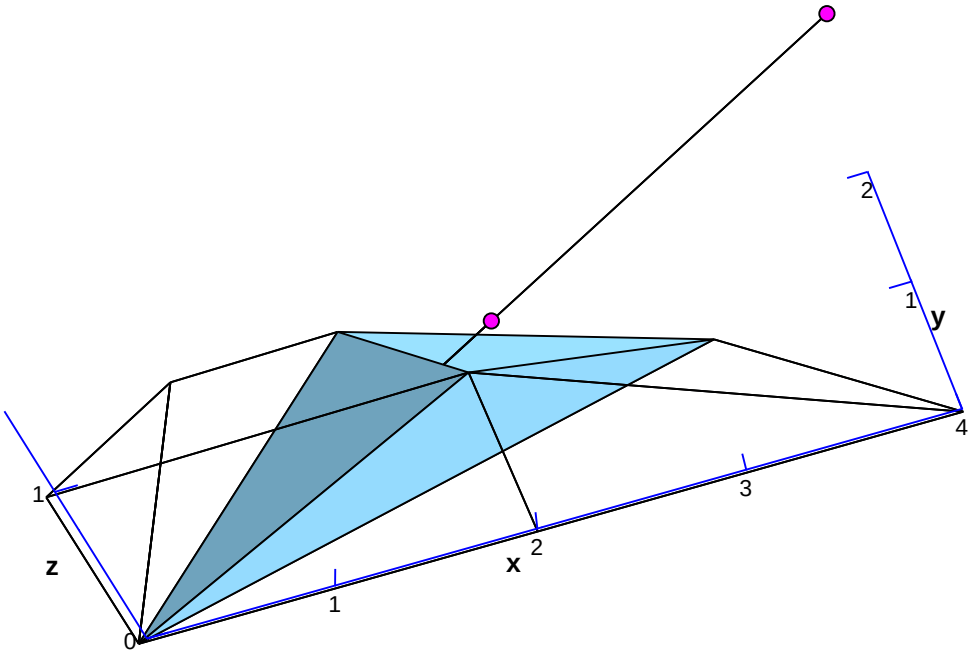


Figure 3.6: Consider the above instance $\tilde{\mathcal{B}}$ with $C = 28$. $\tilde{\mathcal{B}}$ is irreducible according to Definition 3.10. Then, $\mathcal{P}^\star := \operatorname{conv}(\mathbf{0}, (4, 0, 0)^\intercal, (0, 2, 0)^\intercal, (0, 0, 1)^\intercal, (3, 1, 0)^\intercal, (2, 0, 1)^\intercal, (1, 2, 0)^\intercal)$ does not contain $\boldsymbol{v}/2 = (2, 1, 0.5)^\intercal$. Thus, $\mathcal{B}$ is infeasible.

Definition 3.26 (Rational Polyhedral Cone)**.** *A* rational polyhedral cone*, or in short cone $\mathcal{C}$ is denoted by the set of non-negative (real) combinations pos$\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_k\}$ of vectors $\boldsymbol{f}_i \in \mathbb{Z}^m$. A cone is called* pointed *if it does not contain any linear subspace besides the* **0***-space, i.e. if there exists a hyperplane H such that* **0** *is the only element in $H \cap \mathcal{C}$. A cone is called* simplicial*, if all $\boldsymbol{f}_i,\ i \in \{1, \ldots, \ell\}$ are linearly independent.*

Proposition 3.27. *Let $\mathcal{C} = pos\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_k\},\ \boldsymbol{f}_i \in \mathbb{Z}^m$, be a rational polyhedral cone. The set $\mathcal{C} \cap \mathbb{Z}^m$ is finitely generated, that is, there exist $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k \in \mathcal{C} \cap \mathbb{Z}^m$ such that*

$$\mathcal{C} \cap \mathbb{Z}^m = \Big\{ \sum_{i=1}^{n} \lambda_i \boldsymbol{b}_i \ :\ \lambda_i \in \mathbb{N} \Big\}. \tag{3.27}$$

Definition 3.28 (Hilbert Basis)**.** *Let $\mathcal{C}$ be a rational polyhedral cone. A finite generating system of $\mathcal{C} \cap \mathbb{Z}^m$ is called* Hilbert basis *$\mathcal{H}(\mathcal{C})$. A minimal Hilbert basis with respect to inclusion is also called an* integral basis *of the cone $\mathcal{C}$.*

The term 'Hilbert basis' was introduced by GILES & PULLEYBLANK (1979). It was shown by GORDAN (1873) that every rational polyhedral cone has an integral basis. If $\mathcal{C}$ is pointed, it was shown by VAN DER CORPUT (1931) that $\mathcal{H}(\mathcal{C})$ is uniquely determined by

$$\mathcal{H}(\mathcal{C}) = \Big\{ \boldsymbol{b} \in \mathcal{C} \cap \mathbb{Z}^m \setminus \{0\} \,\Big|\, \boldsymbol{b} \neq \boldsymbol{p} + \boldsymbol{q};\ \forall \boldsymbol{p}, \boldsymbol{q} \in \mathcal{C} \cap \mathbb{Z}^m \setminus \{0\} \Big\}. \tag{3.28}$$

In the following context, we assume all cones to be pointed. Furthermore, let m be the dimension of the underlying space.

Definition 3.29 (Closed Convex Lattice Cone)**.** *Let $\mathcal{P}^\star$ be a convex lattice polytope, and $\mathcal{F} = \text{conv}\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_\ell\},\ \boldsymbol{f}_i \in \partial\mathcal{P}^\star,\ \forall i$ be an arbitrary facet of $\mathcal{P}^\star$. By the associated* closed convex lattice cone *$\mathcal{C}(\mathcal{F}) := \text{conv}(\mathbf{0}, \mathcal{F}) \subseteq \mathcal{P}^\star$, we denote the smallest pointed convex cone that contains $\mathcal{F} \subseteq \partial\mathcal{P}^\star$. $\mathcal{F}$ is called* characteristic facet *of $\mathcal{C}(\mathcal{F})$. In particular, $\mathcal{C}(\mathcal{F})$ is* closed *and all of its vertices are integral.*

Definition 3.30 (Nested convex lattice cones)**.** *Let $\mathcal{C}_1 \subset \mathcal{C}_2 \subset \ldots \subset \mathcal{C}_\ell$ be a* nesting *of ℓ n–dimensional pointed convex lattice cones. Each cone $\mathcal{C}_i, i \in \{1, \ldots, \ell\}$ in defined by* **0** *and its characteristic facet $\mathcal{F}_i = \text{conv}\{i\boldsymbol{f}_1, \ldots, i\boldsymbol{f}_\ell\}$. In particular, all $\mathcal{C}_i$ are* closed*. The cones $\mathcal{C}_\ell$ with $\ell \geq 2$ are also called* multiples *of $\mathcal{C}_1$.*

Figure 3.9 shows the construction of $\mathcal{C}_2$ from cones $\mathcal{C}_1$ (cf. Figure 3.9). Analogously, Figure 3.10 constructs $\tilde{\mathcal{P}}_2$ from $\mathcal{C}_1$ which additionally contains vertices dominated by those of $\mathcal{C}_1$.

Considering the search for an optimal solution of (HMBP), the interesting question is now: Given a lattice point $\boldsymbol{v} \in \mathbb{Z}^m_+$ from $\mathcal{C}_\ell$, are there m lattice points $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_m$ from $\mathcal{C}_1$ such that their sum is equal to $\boldsymbol{v}$? Unfortunately, the answer is 'no', at least in dimensions $m > 2$. Basically, this comes from the fact that the integer analogue of Caratheodory's theorem does not hold. BRUNS ET AL. (1999) give an example for that. COOK ET AL. (1986) have shown that – if $\mathcal{H}(\mathcal{C})$ is the integral Hilbert basis of the pointed cone $\mathcal{C}$ – then there exists a subset $\mathcal{H}(\tilde{\mathcal{C}}) \subseteq \mathcal{H}(\mathcal{C})$ of integral vectors $\boldsymbol{b}_i$ from the original Hilbert basis such that

$$\boldsymbol{v} = \sum_{i=1}^{k} \lambda_i \boldsymbol{b}_i,\ \lambda_i \in \mathbb{N},\ \text{and } k \leq 2m - 1, \tag{3.29}$$

where m is the dimension of $\mathcal{C}$. SEBÖ (1990) improved this bound by one to $2m-2$, which is currently the best bound known so far. Whether this bound is tight is still an open question. However, it was shown as well that m is not valid as a bound for dimensions greater than 3 (BRUNS ET AL., 1999). In terms of the HIGH MULTIPLICITY BIN PACKING PROBLEM, the following theorem gives a results for somewhere in between m and $2m - 2$.

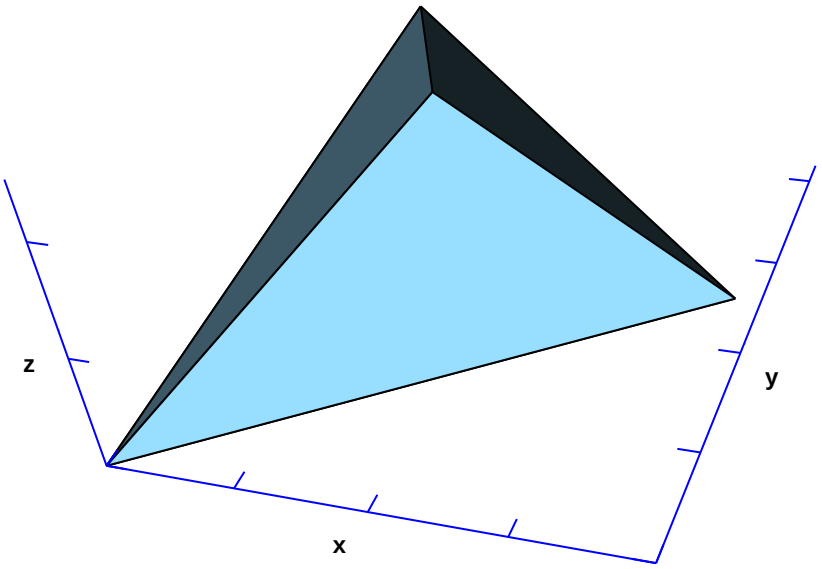


Figure 3.7: The closed convex cone $\mathcal{C}$ is defined by the origin and its characteristic facet $\mathcal{F}$ with the vertices $\boldsymbol{f}_1 = (15, 30, 20)^\intercal$ (on the upmost position) and then counterclockwise $\boldsymbol{f}_2 = (20, 20, 20)^\intercal$ and $\boldsymbol{f}_3 = (40, 20, 5)^\intercal$.

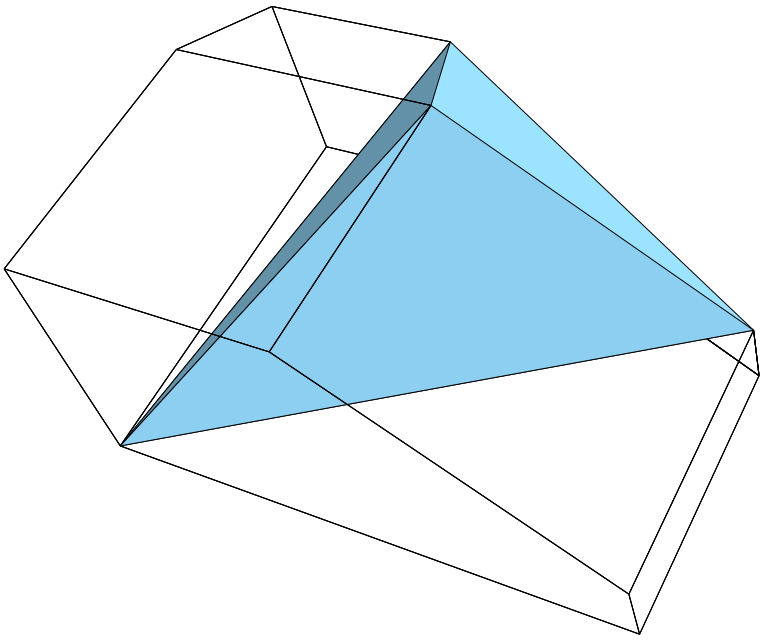


Figure 3.8: The convex lattice cone $\mathcal{C}$ is extended to the lattice polytope $\tilde{\mathcal{P}}$ by adding all lattice points that are dominated by points from $\mathcal{C}$ itself. Geometrically, this corresponds to drawing axis parallel lines from every extremal point $\boldsymbol{f}_i$ from $\mathcal{C}$ to points $\boldsymbol{f}_{ij}, j \in \{1, \ldots, n\}$, where exactly the j-th component is zero. Clearly, lines that are not part of $\partial\tilde{\mathcal{P}}$ might be neglected as it is done in the above figure for lucidity.

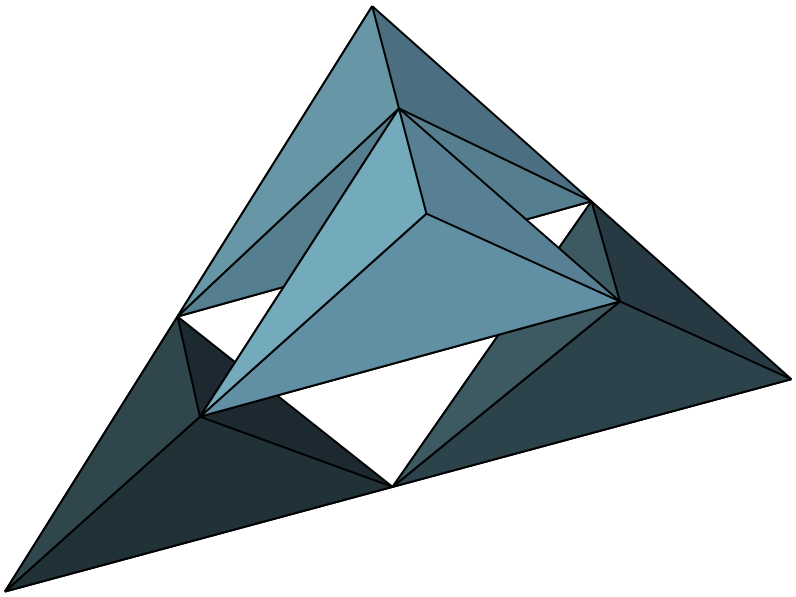


Figure 3.9: The convex lattice cone $\mathcal{C}_2$ is constructed from the cone $\mathcal{C}_1$ from Figure 3.7 by making copies of $\mathcal{C}_1$ and placing them at the positions of the vertices that define its characteristic facet $\mathcal{F}_1$. Clearly, $\mathcal{C}_2$ may contain some vertices in the 'hole' in the upper part between the 3 copies, that cannot be written as the sum of 2 vertices from $\mathcal{F}_1$ or dominated ones. For example consider the point $\boldsymbol{v} = (57, 42, 25)^\intercal$ that is located between the 3 copies of $\mathcal{C}_1$.

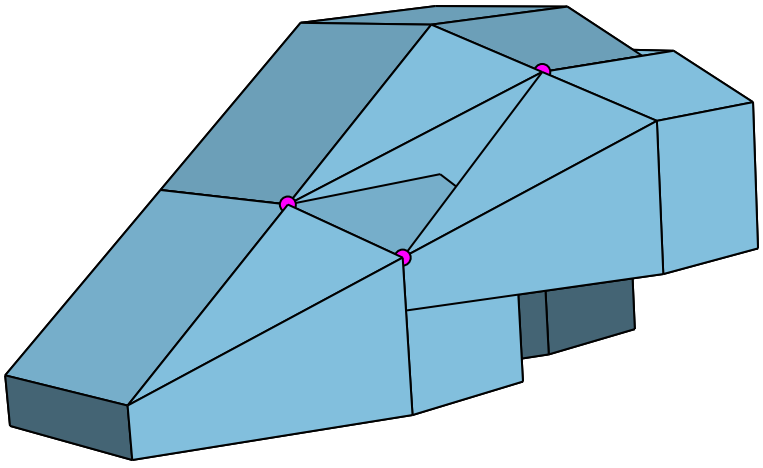


Figure 3.10: The above picture shows $\tilde{\mathcal{P}}_2$ that is constructed from the $\tilde{\mathcal{P}}_1$ from Figure 3.8 by making copies of $\tilde{\mathcal{P}}_1$ and placing them at the positions of the vertices that define its characteristic facet $\mathcal{F}_1$. This picture clarifies the above situation: there might be a (small) convex set within $\tilde{\mathcal{P}}_2$ whose points are not presentable by 2 points of $\tilde{\mathcal{P}}_1$. One facet of this set is tagged by the 3 dotted vertices.

Theorem 3.31 (Solution of (HMBP)). *Let $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C, n)$ be an irreducible instance of* (HMBP) *and $\boldsymbol{v}/n \in \mathcal{P}^\star$. Let $\mathcal{F}_1 := \mathrm{conv}(\boldsymbol{f}_1, \ldots, \boldsymbol{f}_\ell)$ be the facet of the associated lattice polytope $\mathcal{P}^\star$ that is intersected by the straight line through the origin and $\boldsymbol{v}$. Then there is a nesting of cones $\mathcal{C}_1 \subset \mathcal{C}_2 \subset \ldots \subset \mathcal{C}_n$, $\mathcal{C}_i := \mathrm{conv}\{i\boldsymbol{f}_1, \ldots, i\boldsymbol{f}_\ell\}$ with $\boldsymbol{v} \in \mathcal{C}_n$, and $\boldsymbol{v}$ can be represented by at most $\mathcal{O}(m)$ integral vectors chosen from the set of vertices defining $\mathcal{F}_1$, or dominated by these, and an integral residual $\boldsymbol{r} \in \mathcal{P}^\star$.*

Proof. Without loss of generality, assume that $\mathcal{C}_1$ is simplicial. If $\mathcal{C}_1$ is not simplicial, we are always able to find m vertices $\tilde{\boldsymbol{f}}_1, \ldots, \tilde{\boldsymbol{f}}_m$ from $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_\ell\}$ such that their convex hull is intersected by the straight line through $\mathbf{0}$ and $\boldsymbol{v}/n \in \mathrm{conv}(\mathbf{0}, \tilde{\boldsymbol{f}}_1, \ldots, \tilde{\boldsymbol{f}}_m)$. We then choose $\mathcal{C}_1 := \mathrm{conv}(\mathbf{0}, \tilde{\boldsymbol{f}}_1, \ldots, \tilde{\boldsymbol{f}}_m)$. Furthermore, we assume $\boldsymbol{v} \in \mathcal{C}_n \setminus \mathcal{C}_{n-1}$. Otherwise, n might be decreased until it satisfies the assumption. Clearly, if there is a feasible solution using $\tilde{n} < n$ bins, there is a feasible solution using n bins as well. From the above definition of convex lattice cones, it follows in particular that $\mathcal{C}_n \setminus \mathcal{C}_{n-1} = \mathrm{conv}(\mathcal{F}_n, \mathcal{F}_{n-1})$ for $n > 1$.

We prove Theorem 3.31 by induction over n. The induction hypothesis for $n = 1$ is correct: every lattice point $\boldsymbol{v}$ from $\mathcal{C}_1$ is represented by itself. From the definition of $\mathcal{F}_i$, it is obvious that every lattice point located on $\mathcal{F}_{n+1}$ can be translated onto $\mathcal{F}_n$ by a translation vector $\boldsymbol{x} := -\boldsymbol{z}$, where $\boldsymbol{z}$ is an appropriate lattice point on $\mathcal{F}_1$. Therefore, all lattice points located directly on $\mathcal{F}_{n+1}$ might be translated onto $\mathcal{F}_2$ using n integral points from the set of points defining $\mathcal{F}_1$.

As $\mathcal{C}_2 := \mathrm{conv}(\mathbf{0}, \mathcal{F}_2)$ is symmetrically built from cones $\mathcal{C}_1$ (see Fig. 3.9), it suffices to show that any lattice point from $\mathcal{C}_2 \setminus \mathcal{C}_1$ can be translated onto a lattice point from $\mathcal{P}^\star$, which also contains all positive lattice points dominated by those from $\mathcal{C}_1$ in addition (see Fig. 3.8).

So consider a point $\boldsymbol{r} \in (\mathcal{C}_2 \setminus \mathcal{C}_1) \cap \mathbb{Z}^m$. By construction, it is $\boldsymbol{r} \in \mathrm{conv}(\boldsymbol{f}_1, \ldots, \boldsymbol{f}_n, 2\boldsymbol{f}_1, \ldots, 2\boldsymbol{f}_n)$. Geometrically, $\mathcal{C}_2$ is constructed by making n copies of $\mathcal{C}_1$ and placing them with their origins onto the lattice points $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_n$ from $\mathcal{F}_1$. Clearly, if $\boldsymbol{r}$ is located within one of the translated copies of $\mathcal{C}_1$ originating at $\boldsymbol{f}_\ell$, it can be easily transformed onto $\mathcal{F}_1 \subset \mathcal{P}^\star$ by a translation vector $\boldsymbol{t} := \boldsymbol{r} - \boldsymbol{f}_i$, $\boldsymbol{f}_i \in \mathcal{F}_1 \cap \mathbb{Z}^m$. Clearly, $\boldsymbol{t} \in \mathcal{C}_1 \subseteq \mathcal{P}^\star$. If $\boldsymbol{r}$ is dominated by any lattice point from the translated copies mentioned above, we are able to choose a point $\boldsymbol{t} \in \mathcal{P}^\star$ analogously. Note that $\boldsymbol{t}$ is not necessarily chosen from $\mathcal{C}_1$.

Now, assume that $\boldsymbol{r}$ is not located within one of the translated copies and not dominated by any of their lattice points. W.l.o.g., we assume $\boldsymbol{r}$ to be maximal, so choose an integral vector $\boldsymbol{r}$ as a convex combination from $\{2\boldsymbol{f}_1, \ldots, 2\boldsymbol{f}_\ell\}$, that is $\boldsymbol{r} \in \mathcal{F}_2$. So let $\boldsymbol{r} = \sum_{i=1}^{\ell} \alpha_i \cdot 2\boldsymbol{f}_i$, $\sum_{i=1}^{\ell} \alpha_i = 1$, $\boldsymbol{r} \in \mathbb{Z}^m$. Considering the original (HMBP) instance $\mathcal{B}$, $\boldsymbol{r}$ induces a weight

$$w(\boldsymbol{r}) := \boldsymbol{r}^\intercal \boldsymbol{w} = \sum_{i=1}^{\ell} w(\alpha_i \cdot 2\boldsymbol{f}_i) = 2\sum_{i=1}^{\ell} w(\alpha_i \boldsymbol{f}_i) \leq 2C, \sum_{i=1}^{\ell} \alpha_i = 1. \tag{3.30}$$

Clearly, $\boldsymbol{r}$ is feasible with respect to the capacity bound $2C$, but not necessarily representable by 2 patterns from the lattice cone $\mathcal{C}_1$. Therefore, we show that there is always a point $\boldsymbol{d} := \boldsymbol{f}_p + \boldsymbol{f}_q \in \mathcal{F}_2$ with $\boldsymbol{f}_p + \boldsymbol{f}_q \in \mathcal{F}_1$ and a vector $\boldsymbol{t} := \boldsymbol{d} - \boldsymbol{r}$ such that $\boldsymbol{t}^+ \in \mathcal{P}^\star$, and $w(\boldsymbol{t}^+) \leq C$. For this reason, let

$$\boldsymbol{t}^+ := (t_1^+, \ldots, t_m^+), \text{ where } t_\ell^+ := \begin{cases} t_\ell, \text{ if } t_\ell > 0 \\ 0, \text{ otherwise.} \end{cases} \tag{3.31}$$

The operator '+' simply projects negative components of a vector to zero. Note that after the projection of negative components to zero, the resulting vector $\boldsymbol{t}^+$ has decreased in length. This can be easily verified by repeatedly applying the Theorem of Pythagoras, for each projected component once. Moreover, note that it is always possible to subtract the components of the pattern $\boldsymbol{t}^+ - \boldsymbol{t}$ resulting

from the projection from other patterns chosen in a solution to the instance $\mathcal{B}$. This is due to the fact that $\boldsymbol{f}_\ell \geq 0$, for all $\ell \in \{1,\ldots,m\}$, and the sum of all patterns used in a solution equals $\boldsymbol{v}$ which is nonnegative. Therefore, $\boldsymbol{t}^+ - \boldsymbol{t} \leq \boldsymbol{v}$. More demonstratively, if the translation by $\boldsymbol{t}$ amounts to a translation onto a vertex $\boldsymbol{d}$ that is the representable by the sum of two vertices $\boldsymbol{f}_p, \boldsymbol{f}_q \in \mathcal{F}_1$, then the translation by $\boldsymbol{t}^+$ amounts to a translation onto a vertex dominated by $\boldsymbol{d}$.

We show now, that all translations $\boldsymbol{t}_\ell$ between extremal points of $\mathcal{C}_1$ are feasible bin patterns with respect to $w(\boldsymbol{t}_\ell^+)$. For all translations given by $\boldsymbol{f}_1,\ldots,\boldsymbol{f}_m$ it is by definition $w(\boldsymbol{f}_\ell^+) = w(\boldsymbol{f}_\ell) \leq C$, $\ell \in \{1,\ldots,m\}$. Now regard the remaining edges of $\mathcal{C}_1$ given by $\boldsymbol{f}_{pq} := \boldsymbol{f}_p - \boldsymbol{f}_q$, for $p,q \in \{1,\ldots,m\}$. It is easy to see that

$$w(\boldsymbol{f}_{pq}^+) \;\geq\; w(\boldsymbol{f}_p - \boldsymbol{f}_q) \;=\; w(\boldsymbol{f}_p) - w(\boldsymbol{f}_q) \;\leq\; C. \tag{3.32}$$

For a contradiction, we assume that $w(\boldsymbol{f}_{pq}^+) > C$. In order to maximize $w(\boldsymbol{f}_{pq}^+)$, we have to choose $\boldsymbol{f}_p \geq \boldsymbol{f}_q$. Clearly, as $\boldsymbol{f}_p, \boldsymbol{f}_q$ are nonnegative, the maximal weight can be obtained by setting $\boldsymbol{f}_q = \boldsymbol{0}$. Then, we have $w(\boldsymbol{f}_p) - w(\boldsymbol{0}) = \boldsymbol{f}_p > C$, which is a contradiction to the definition of $\boldsymbol{f}_p$.
From the fact that $\mathcal{C}_1, \mathcal{C}_2$ are simplicial, it follows that $\mathcal{F}_1, \mathcal{F}_2$ are simplicial, too. As $\mathcal{C}_2$ is constructed from simplicial cones $\mathcal{C}_1$, $\mathcal{F}_2$ can be canonically partitioned into simplices congruent to $\mathcal{F}_1$. In the above partition, all appearing edges $\boldsymbol{f}_p, \boldsymbol{f}_q, \boldsymbol{f}_{pq}$, $p,q \in \{1,\ldots,m\}$ are parallel. Therefore, the dilatation of these simplices with respect to the coordinates is identical for each dimension. Hence, there is always at least one from the above translation vectors (the edges of the simplex) that projects any point of any simplex within $\mathcal{C}_2$ onto a point located either in $\mathcal{C}_1$ itself, or in one of the translated copies of $\mathcal{C}_1$ within $\mathcal{C}_2$.

Consider again the vertex $\boldsymbol{r}$ on facet $\mathcal{F}_2$. Clearly, $\boldsymbol{r}$ is located within at least one simplex that is congruent to $\mathcal{F}_1$. We have shown above that all extremal moves translating vertices of $\mathcal{C}_1$ into each other are feasible bin patterns. As $\boldsymbol{r}$ is located within a simplex that is defined by these vertices, we are always able to find at least one integral vertex $\boldsymbol{d} := \boldsymbol{f}_p + \boldsymbol{f}_q \in \mathcal{F}_2$ with $\boldsymbol{f}_p + \boldsymbol{f}_q \in \mathcal{F}_1$ and a vector $\boldsymbol{t} := \boldsymbol{d} - \boldsymbol{r}$ such that $w(\boldsymbol{t}^+) \leq C$. It follows that $\boldsymbol{t}^+ \in \mathcal{P}^\star$. □

The Figures 3.7 – 3.15 illustrate the construction of cones $\mathcal{C}_1, \mathcal{C}_2$ and polytopes $\mathcal{P}^\star = \tilde{\mathcal{P}}$ and $\tilde{\mathcal{P}}_2$. Figure 3.15 clearly shows that there might be vertices that are located in a 'hole' within $\mathcal{C}_2$ that is not covered by vertices from translated copies of $\mathcal{C}_1$ or dominated points. Figures 3.11 – 3.14 show the construction of a solution to a (HMBP) instance of dimension 3 which has no trivial solution.

Corollary 3.32 (Solution with $n+1$ bins)**.** *Given an irreducible instance $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C, n)$ of* (HMBP)*. If $\boldsymbol{v}/n$ is part of the characteristic polytope $\mathcal{P}^\star$, then there is a solution using at most $n+1$ bins. Any solution has size $\mathcal{O}(m^2)$.*

Proof. From Theorem 3.31, we know that $\boldsymbol{v}$ can be decomposed into n integral vectors from $\mathcal{F}_1$ and a residual $\boldsymbol{r} \in \mathcal{P}^\star$. As $\mathcal{F}_1$ is defined by at most m vertices, it follows that there are in total $\mathcal{O}(m)$ pairwise different patterns used in a solution including dominated patterns resulting from negative components in the residual $\boldsymbol{r}$. Each pattern has dimension m, so the size of a solution is $\mathcal{O}(m^2)$ in total. □

Note that the number of pairwise different bin patterns used in a solution may vary from 1 to $2m$. The first case occurs if $\boldsymbol{v}$ can be simply represented by n patterns $\boldsymbol{v}/n$. The second case occurs if we are given $\boldsymbol{v} = (v_1,\ldots,v_m)^\intercal$, with $v_\ell \geq 2$ for all $\ell \in \{1, ldots, m\}$, and facet defining vertices $\boldsymbol{f}_1 = (v_1 - 1, 0, \ldots, 0)^\intercal$, $\boldsymbol{f}_2 = (0, v_2 - 1, 0, \ldots, 0)^\intercal$, …, $\boldsymbol{f}_m = (0, 0, \ldots, v_m - 1)^\intercal$. Clearly, in a solution we need m original patterns $\boldsymbol{f}_\ell$, $\ell \in \{1,\ldots,m\}$, and m patterns $(1,0,\ldots,0)^\intercal$, $(0,1,\ldots,0)^\intercal$, …, $(0,\ldots,0,1)^\intercal$, dominated by those. Considering non pathologic instances, the number of used patterns is usually less than $m+2$, that is at most m patterns induced by the facet defining vertices,

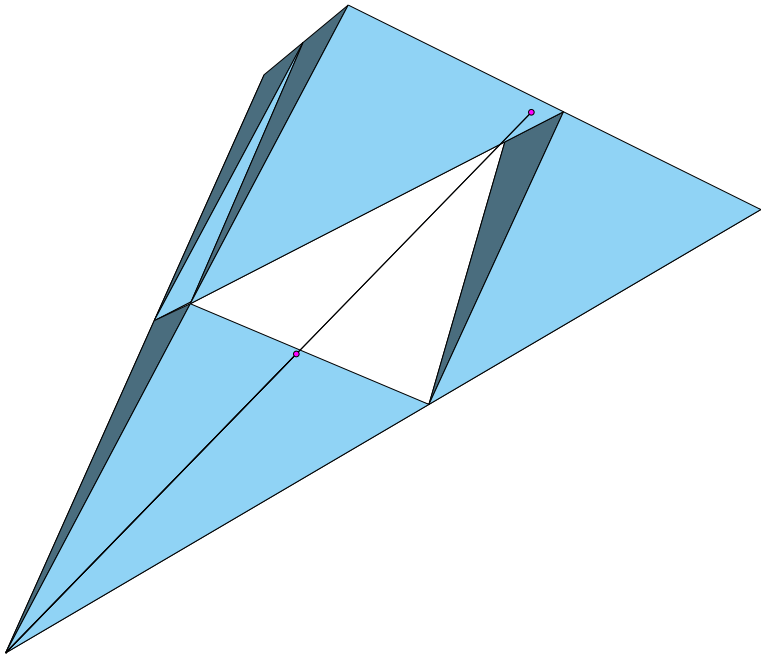


Figure 3.11: Feasible (HMBP) instance $\mathcal{B}$ of dimension 3. $\mathcal{C}_1$ is constructed by $\mathbf{0}$, $\boldsymbol{f}_1 = (15, 30, 15)^\intercal$, $\boldsymbol{f}_2 = (20, 20, 20)^\intercal$, and $\boldsymbol{f}_3 = (40, 20, 5)^\intercal$. The points $\boldsymbol{v} = (57, 42, 25)^\intercal$ and $\boldsymbol{v}/2 = (28.5, 21, 12.5)^\intercal$ are depicted by the small red dots in the illustration.

.

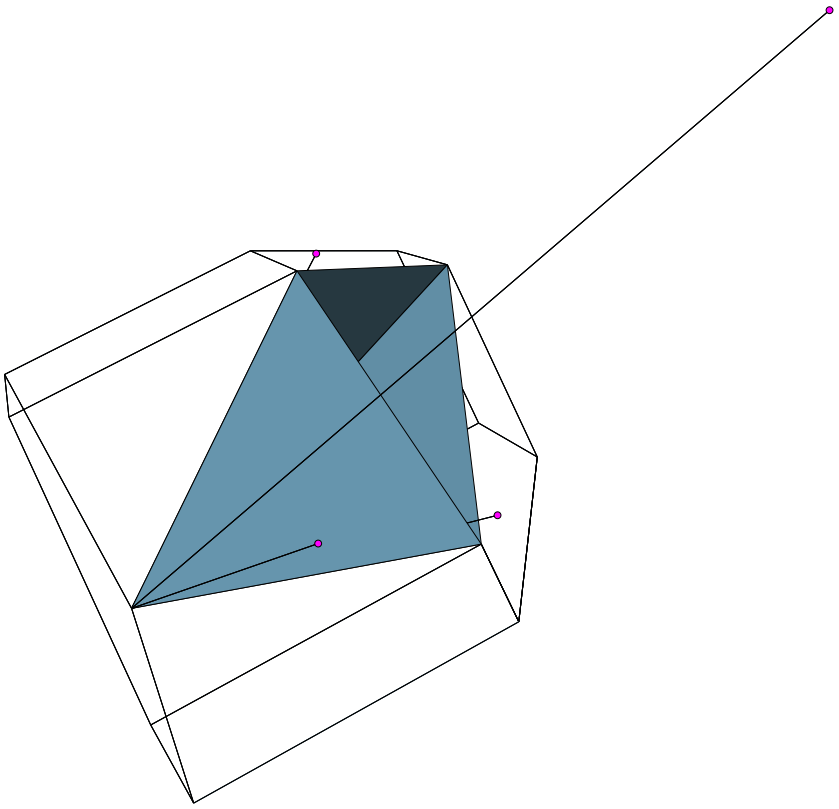


Figure 3.12: $\mathcal{C}_1$ is extended by points dominated by $\boldsymbol{f}_1$, $\boldsymbol{f}_2$, and $\boldsymbol{f}_3$. Subtracting either $\boldsymbol{f}_1$, or $\boldsymbol{f}_2$, or $\boldsymbol{f}_3$ from $\boldsymbol{v}$ does not result in a vertex located in the extension of $\mathcal{C}_1$. $\boldsymbol{v}$ and the resulting vertices after subtraction are depicted by the small red dots in the illustration. Clearly, there cannot be any solution using only 2 patterns from $\{\mathbf{0}, \boldsymbol{f}_1, \boldsymbol{f}_2, \boldsymbol{f}_3\}$

.

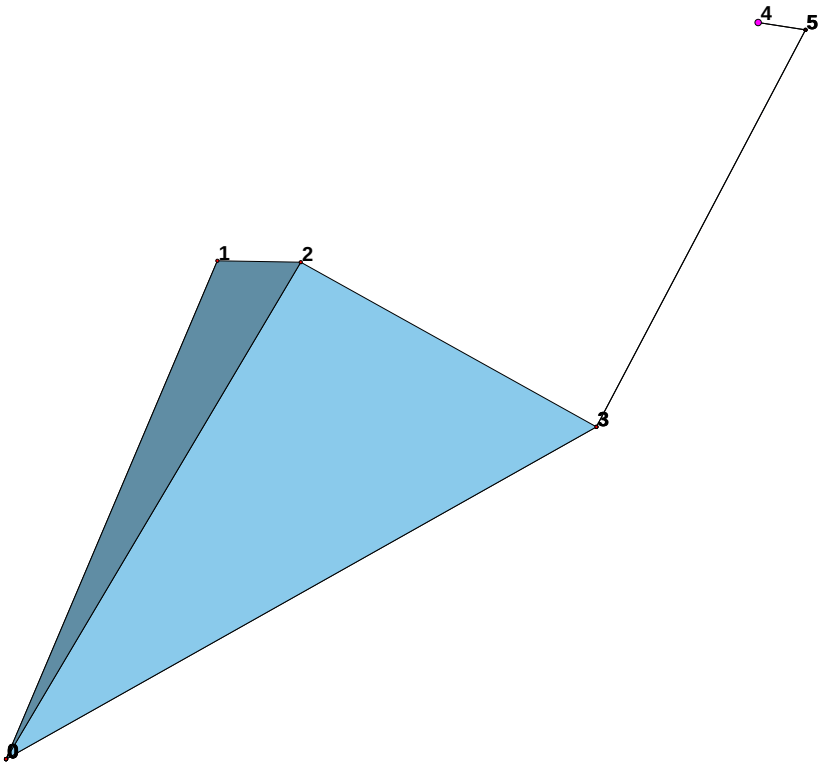


Figure 3.13: Construction of a solution to $\mathcal{B}$ using 3 patterns. $\boldsymbol{v}$ has label 4. For a solution, we use patterns $\boldsymbol{f}_3$ (label 3), $\boldsymbol{f}_2$ (the edge from 3 to 5), and a residual $\boldsymbol{r} = (-3, 2, 0)^\intercal$ (the edge from 5 to 4).

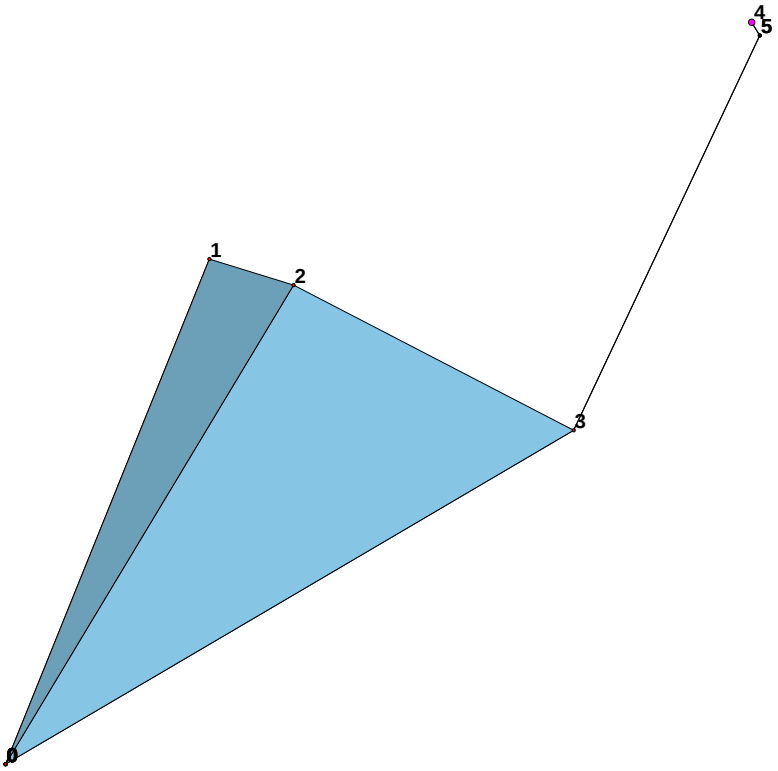


Figure 3.14: Negative components of $\boldsymbol{r}$ are subtracted from other patterns. In this example, the negative component vector $(-3, 0, 0)^\intercal$ from the residual $\boldsymbol{r}$ is subtracted from $\boldsymbol{f}_2$. Hence, a solution is constructed by $\boldsymbol{f}_3$, $\boldsymbol{f}_2 - (-3, 0, 0)^\intercal$, and $(0, 2, 0)^\intercal$.

one residual pattern, and one dominated pattern arising from the subtraction of negative components of the residual as shown in the proof of Theorem 3.31.

Graphically speaking, in Theorem 3.31, we consider the "holes" on the facet $\mathcal{F}_2$, these are the regions that are not covered by copies from $\mathcal{F}_1$ (see Fig. 3.9 and Fig. 3.15 for an example in dimension 3). A linear translation of these "holes" down to $\mathcal{P}^\star$ by a vector $-\boldsymbol{f}$, $\boldsymbol{f} \in \{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m\}$ is not necessarily located within $\mathcal{P}^\star$. Therefore, there might be integral points within $\mathcal{C}_2$ that are not representable by 2 points from $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m\}$ or dominated by those. With one additional point from $\mathcal{P}^\star$, these are covered. We have shown that there is always a feasible pattern from $\mathcal{P}^\star$ that satisfies the requirement. Graphically speaking, this is due to the fact that the axial extensions of $\mathcal{P}^\star$ are defined by the facet defining vertices $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m\}$ of the cone $\mathcal{C}_1$. Therefore, all vectors $\boldsymbol{t}^+ := (\boldsymbol{f}_p - \boldsymbol{f}_q)^+$, $\boldsymbol{f}_p, \boldsymbol{f}_q \in \{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m\}$ with negative entries mapped to zero are in $\mathcal{P}^\star$.

In case $\boldsymbol{t}$ contains negative entries, those might be subtracted from other vectors chosen from the set of vertices defining $\mathcal{F}_1$ in a solution. The resulting vectors are then dominated by vectors from $\mathcal{F}_1$. If all components from $\boldsymbol{r}$ are negative or zero, there is a solution with n bins. By the above procedure, we are always guaranteed to find a solution with $n+1$ bins at most. But even then there are good chances to construct a solution with only n bins.

One idea to provably achieve this, is to increase the number of vertices defining $\mathcal{F}_1$. If $\mathcal{F}_1$ is defined by an $(m-1)$–dimensional parallelepiped consisting of 2^{m-1} vertices, the multiples of $\mathcal{F}_1$ would define facets $\mathcal{F}_2, \ldots, \mathcal{F}_\ell$ which do not contain any 'holes' as in the above case. The Figures 3.15 and 3.16 illustrate this for the 3–dimensional case. Clearly, all vertices within a cone $\mathcal{C}_\ell$ would then be representable by vertices from $\mathcal{C}_1$ or dominated ones. Practically, this approach is difficult to realize as an exponential number of facet defining vertices is needed. Therefore, we will present some other approaches towards an optimal solution.

Corollary 3.33 (Solution with n bins)**.** *Given an irreducible instance $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C, n)$ of* (HMBP) *and $\boldsymbol{v}/n$ is part of the characteristic polytope $\mathcal{P}^\star$. If we can find vertices $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m$ spanning a facet $\mathcal{F}_1$ such that $\boldsymbol{v} \in \mathcal{C}_n$ is located within a copy of $\mathcal{C}_1 := \operatorname{conv}(\mathbf{0}, \mathcal{F}_1)$ or dominated by one of its vertices, there is a solution with no more than n bins using at most $m+1$ pairwise different patterns from $\mathcal{P}^\star$.*

Proof. See Proof of Theorem 3.31 for the construction of a solution. For maximality, we assume $n > m$. Every single pattern from $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m\}$ might be used (repeatedly) in order to represent $\boldsymbol{v} - \boldsymbol{r}$. Clearly, the residual $\boldsymbol{r}$ is in $\mathcal{P}^\star$, but not necessarily from the set $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m\}$. □

Corollary 3.34 (Solution with n bins)**.** *Given an irreducible instance $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C, n)$ of* (HMBP) *and $\boldsymbol{v}/n$ is part of the characteristic polytope $\mathcal{P}^\star$. If we can find vertices $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m$ spanning a facet $\mathcal{F}_1$ such that $\boldsymbol{v}/n \in \mathcal{C}_1 := \operatorname{conv}(\mathbf{0}, \mathcal{F}_1)$, and*

$$\boldsymbol{v} - \sum_{i=1}^{n} \boldsymbol{f}_{\ell_i} \leq \mathbf{0},\ \boldsymbol{f}_{\ell_i} \in \{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m\}, \tag{3.33}$$

there is a solution with no more than n bins.

Proof. We use exactly n patterns from $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m\}$ in a solution, and the residual is negative, which means that we might subtract the excessive items from appropriate patterns. These always remain feasible as they are dominated by the original patterns. □

Corollary 3.35 (Solution with n bins)**.** *Given an irreducible instance $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C, n)$ of* (HMBP)*, a facet $\mathcal{F} = \operatorname{conv}(\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m) \subset \mathcal{P}^\star$ that is intersected by the straight line connecting $\mathbf{0}$ and $\boldsymbol{v}$, and an integer point $\boldsymbol{f}_{m+1} \in \mathcal{P}^\star$ such that the simplex $\operatorname{conv}(\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m, \boldsymbol{f}_{m+1})$ is unimodular. Then $\mathcal{B}$ is feasible and there is a solution consisting of at most n vectors from $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_{m+1}\}$.*

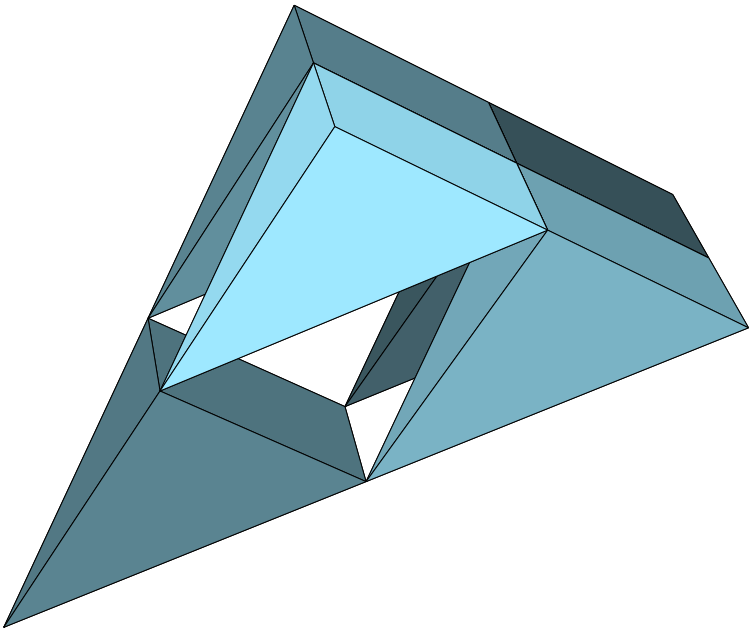


Figure 3.15: $\mathcal{C}_2$ is constructed from the cone $\mathcal{C}_1$ as above, but this time using 4 vertices. Clearly, $\mathcal{C}_2$ may not contain vertices that are not covered by dominating vertices from $\mathcal{C}_2$, because the face $\mathcal{F}_2$ defining $\mathcal{C}_2$ does not contain any holes.

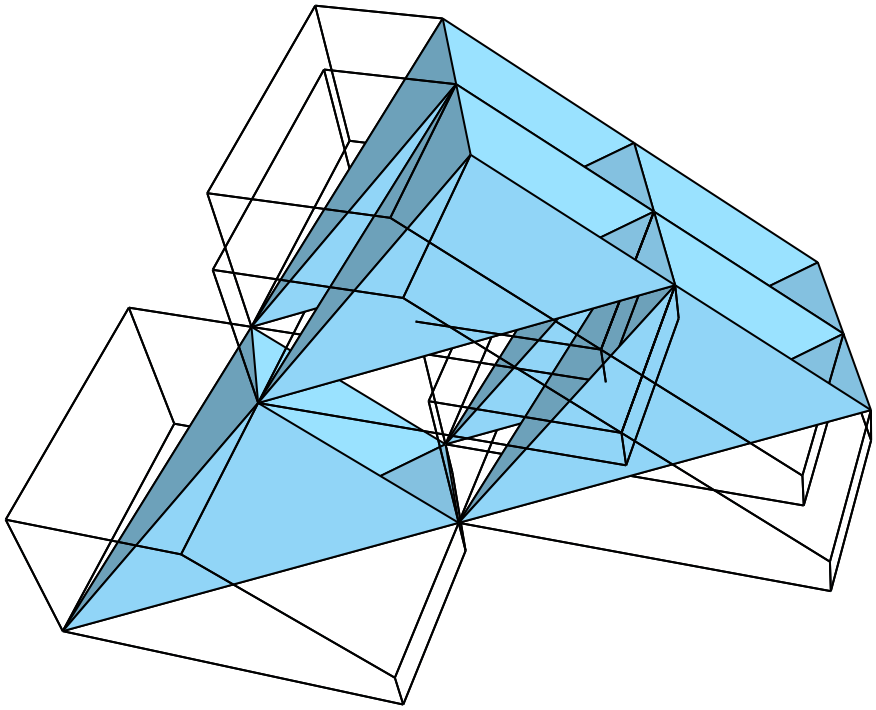


Figure 3.16: The above construction shoes $\tilde{\mathcal{P}}_2$ that is built from $\tilde{\mathcal{P}}_1$. Clearly, all integral vertices in the lattice cone $\mathcal{C}_2$ are either covered or dominated by the 4 copies from $\tilde{\mathcal{P}}_1$.

Proof. From Lemma 3.12, we know $\mathcal{B}$ is feasible. If we can find feasible patterns $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_{m+1}$ such that the simplex $\mathcal{S} := \operatorname{conv}(\mathbf{0}, \boldsymbol{f}_1, \ldots, \boldsymbol{f}_{m+1})$ is unimodular, we might construct a feasible solution from vectors $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_{m+1}\}$ according to the proof of Lemma 3.12. □

3.4.3 Computation of relevant bin patterns

In general, the associated lattice polytope $\mathcal{P}^\star$ of an instance $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C, n)$ is neither explicitly given, nor can it be computed in polynomial time. Therefore, the search for a corresponding facet $\mathcal{F}$ of $\mathcal{P}^\star$ that is intersected by the straight line connecting $\boldsymbol{v}$ and $\mathbf{0}$, might be a bold venture.

But things are a little bit easier: from Theorem 3.31, we know that any cone $\mathcal{C}$ would be convenient as long as it is located completely within $\mathcal{P}^\star$ and it contains $\boldsymbol{v}/n$. Therefore the quest is the following: Find at most m vertices $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m\}$ in $\mathbb{Z}^m_+$ satisfying the following conditions:

- $\boldsymbol{v}/n \in \operatorname{conv}(\mathbf{0}, \boldsymbol{f}_1, \ldots, \boldsymbol{f}_m)$,
- $\boldsymbol{v}^\intercal \boldsymbol{w}/n \leq \boldsymbol{f}_i^\intercal \boldsymbol{w} \leq \tilde{C},$ for at least some $i \in \{1, \ldots, m\},$ if all $\boldsymbol{f}_i$ are located on the same facet, or alternatively
- $\boldsymbol{v}^\intercal \boldsymbol{w}/n \leq \boldsymbol{f}_i^\intercal \boldsymbol{w} \leq \tilde{C},$ for all $i \in \{1, \ldots, m\}$, where

$$\tilde{C} := \max\left\{\boldsymbol{x}^\intercal \boldsymbol{w} \;\middle|\; \boldsymbol{x} \in \{v_1, 0, \ldots, 0\} \times \ldots \times \{0, \ldots, 0, v_m\},\ \boldsymbol{x}^\intercal \boldsymbol{w} \leq C\right\},$$

  in both cases, which means $\mathcal{B}$ is irreducible according to Section 3.1.4,
- all $\boldsymbol{f}_i$ are pairwise linearly independent.

In this section, we will present two approaches to efficiently compute the bin patterns that are necessary to generate a solution motivated by the theoretical considerations from the preceding section.

Computation by a Mixed Integer Quadratic Program (MIQP)

The computation of m facet defining vertices $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m$ amounts to solving the following Mixed Integer Quadratic Program (MIQP):

$$
\begin{aligned}
(MIQP) \qquad & \text{maximize } \boldsymbol{v}x && && (3.34)\\
\text{s.t.} \quad & \sum_{i=1}^{m} \lambda_i f_{ij} = x v_j, && \forall j \in \{1, \ldots, m\}, && (3.35)\\
& \sum_{i=1}^{m} \lambda_i = 1, && && (3.36)\\
& \boldsymbol{f}_i^\intercal \boldsymbol{w} \leq \tilde{C}, && \forall i \in \{1, \ldots, m\}, && (3.37)\\
& \lambda_i \geq 0, && \forall i \in \{1, \ldots, m\}, && (3.38)\\
& x \geq 0, && && (3.39)\\
& \boldsymbol{f}_i \in \mathbb{Z}^m_+, && \forall i \in \{1, \ldots, m\}. && (3.40)
\end{aligned}
$$

Note that the above program is linear except for constraint (3.35) which is quadratic. We are looking for m vertices defining a facet that is intersected in the point $x\boldsymbol{v}$ by the straight line interconnecting

$\mathbf{0}$ and $\boldsymbol{v}$ (cf. equations (3.35), (3.36) and (3.36)). At the same time, we maximize $\boldsymbol{v}x$, that is we require the point $x\boldsymbol{v}$ to have maximum distance from the origin such that the above conditions are satisfied. Constraint (3.37) requires all vertices $\boldsymbol{f}_\ell$ to be feasible bin patterns. (3.39) requires x to be non-negative, and (3.40) all vertices $\boldsymbol{f}_\ell$ to be integral.

The MIQP that is stated above is quite small. It might be solved by a standard Quadratic Programming technique like for example an interior point method. As all variables are integral and nonnegative (except for x which might be fractional), these algorithms usually compute an optimal solution in polynomial time.

Computation if $\mathcal{P}^\star$ is (partially) given

Due to the fact that it might not seem very skillful to solve a Mixed Integer Linear Program like (HMBP) by means of solving a Quadratic Program, we present another approach that is based on theoretical insights from GRÖTSCHEL ET AL. (1993) and EISENBRAND (2003). This strategy has been originally proposed by AGNETIS & FILIPPI (2005) who proceed on the assumption that we have given an explicit representation of the lattice polytope $\mathcal{P}^\star$ corresponding to $\mathcal{B}$. This is practically not useful, because an enumeration of lattice points of $\mathcal{P}^\star$ cannot be achieved in polynomial time at all. Therefore, we will extend this to an approach with a more practical orientation later on by restricting the size of the point set to consider. But first of all, we will show how it works in general.

Given an explicit representation of the lattice polytope $\mathcal{P}^\star$ corresponding to $\mathcal{B}$, the facet defining vertices $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m$ might be computed using the following primal–dual programs (P) and (D). In the following, let $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_k\}$ be the set of vertices defining $\mathcal{P}^\star$.

$$(P) \qquad \text{maximize } x \tag{3.41}$$

$$\text{s.t.} \quad \sum_{i=1}^{k} \lambda_i p_{ij} = x v_j, \qquad \forall j \in \{1, \ldots, m\}, \tag{3.42}$$

$$\sum_{i=1}^{k} \lambda_i = 1, \tag{3.43}$$

$$\lambda_i \geq 0, \qquad \forall i \in \{1, \ldots, k\}, \tag{3.44}$$

$$x \geq 0, \tag{3.45}$$

Informally speaking, we search for facet defining vertices of $\mathcal{P}^\star$, which contain the point $x\boldsymbol{v}$ on the straight line connecting $\mathbf{0}$ and $\boldsymbol{v}$ farthest from the origin in their convex hull. The corresponding dual program is:

$$(D) \qquad \text{minimize } y \tag{3.46}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} \sigma_j p_{ij} + y \geq 0, \qquad \forall i \in \{1, \ldots, k\}, \tag{3.47}$$

$$\sum_{j=1}^{m} \sigma_j v_j \leq -1. \tag{3.48}$$

Clearly, (P) and (D) are feasible, and a nontrivial optimal basis solution exists if $x > 0$. Any nontrivial basis solution of (P) is represented by at most m vertices $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m$ from $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_k\}$. Note that $-\big(\sum_{j=1}^{m} \sigma_j p_{ij} + y\big)$ represent the reduced cost coefficients for the primal variables λ_i, $i \in \{1, \ldots, k\}$.

An optimal basis of (P) can be found in strongly polynomial time using the ellipsoid method. This is due to a result in GRÖTSCHEL ET AL. (1993) Section 6.6, in particular Theorem 6.6.5. Among other things, it states that for any well-described polyhedron specified by a strong separation oracle that can be evaluated in polynomial time, there is a polynomial algorithm that finds a basic optimum standard dual solution if one exists.

The *separation problem* mentioned above consists of the decision whether a vertex $(\sigma_1, \ldots, \sigma_m, y) \in \mathbb{Z}^{m+1}$ is feasible or not. This can be achieved by solving the following KNAPSACK PROBLEM in fixed dimension:

$$
\begin{aligned}
(KP) \qquad & \text{maximize } \boldsymbol{\sigma}^\intercal \boldsymbol{x} && (3.49)\\
\text{s.t.} \quad & \boldsymbol{w}^\intercal \boldsymbol{x} \leq C, && (3.50)\\
& \boldsymbol{x} \in \mathbb{Z}_+^m, && (3.51)
\end{aligned}
$$

Clearly, if the objective function value of (KP) is greater or equal $-y$, the vertex $(\sigma_1, \ldots, \sigma_m, y) \in \mathbb{Z}^{m+1}$ is feasible with respect to the dual solution space.

Using a preprocessing technique presented by FRANK & TARDOS (1987), the separation problem (KP) can be transformed in constant time into an equivalent problem of size polynomial in m. Using an algorithm for integer programming in fixed dimension due to EISENBRAND (2003), the separation problem can be solved in $\mathcal{O}(\log u_{\max})$. It follows from Theorem 6.6.5 in GRÖTSCHEL ET AL. (1993) that we are able to find a feasible dual basis solution for (D), and hence an optimal basis for (P) in polynomial time.

As mentioned above, an explicit representation of $\mathcal{P}^\star$ is not available and not computable within polynomial time. For this reason, our considerations are slightly different: in order to obtain an optimal basis for (P), only that part of $\mathcal{P}^\star$ is explicitly needed that contains the facet $\mathcal{F}_1$. Hence, we can narrow the set $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_k\}$ used in (P) and (D) to a much smaller subset $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_\ell\}$. The question is how to choose these vertices efficiently. The worst thing that happens if we chose the 'wrong' vertices, is that (P) has the trivial solution $x = 0$, and we have to start over with another vertex set.

The choice of vertices $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_\ell\}$ is motivated heuristically by generating $\ell \geq m$ integral points in the neighborhood of $\boldsymbol{v}/n$. For this purpose, regard the points $\lfloor \boldsymbol{v}/n \rfloor$ and $\lceil \boldsymbol{v}/n \rceil$. Usually, $\lceil \boldsymbol{v}/n \rceil$ is not in $\mathcal{P}^\star$. If this is the case even so, choosing n patterns $\lceil \boldsymbol{v}/n \rceil$ yields a point dominating $\boldsymbol{v}$. So we only have to subtract the pattern $n \cdot \lceil \boldsymbol{v}/n \rceil - \boldsymbol{v}$ from some of the n patterns $\lceil \boldsymbol{v}/n \rceil$ in order to obtain a solution to the original instance. So assume $\lceil \boldsymbol{v}/n \rceil \notin \mathcal{P}^\star$. We know that $\lfloor \boldsymbol{v}/n \rfloor \in \mathcal{P}^\star$, because $w(\lfloor \boldsymbol{v}/n \rfloor) \leq w(\boldsymbol{v}) \leq C$. So, we start with $\boldsymbol{p}_k := \lfloor \boldsymbol{v}/n \rfloor$ for $k \in \{1, \ldots, m\}$, and increase for each k exactly the k-th component p_{kk} to the maximum possible increment such that the weight of the resulting vector still satisfies the capacity bound C. Clearly, p_{kk} must also satisfy $p_{kk} \leq u_k$, with u_k being an upper bound on the number of items of type k packed into a single bin according to its definition in (3.13). It might happen that we cannot increase each component by at least one. In order to do this anyhow, we have to decrease other components. The choice of the components is surely motivated by the best possible exploitation of the bin capacity. Naturally, the search could also be started from $\lceil \boldsymbol{v}/n \rceil$ decreasing single components until they are feasible with respect to C and (3.13).

Clearly, the presented method resembles only a naive approach. More sophisticated methods are to develop, if the above strategy does not yield to relevant patterns with respect to a nontrivial solution of the primal program from above. The considerations from Section 3.2.3 might be helpful doing this. For example, we might consider the gap between $w(\lfloor \boldsymbol{v}/n \rfloor)$ and C to compute offsets $\boldsymbol{o}_k$ which minimize $C - (w(\lfloor \boldsymbol{v}/n \rfloor) - w(\boldsymbol{o}))$. To bound complexity, each $\boldsymbol{o}_{k\ell}$ has to be located within a given

interval $[-\bar{o}_\ell, \bar{o}_\ell]$ which depends on the size and multiplicity of item ℓ. This amounts to implicitly taking into account the orientation of the hyperplane defined by the capacity constraint $\boldsymbol{u}^\intercal \boldsymbol{w} \leq \tilde{C}$ which separates feasibly from infeasible patterns. For an efficient implementation of the pattern search, we may use the concept of heteroary codings from Section 3.2.2.

3.4.4 Computation of (HMBP) solutions

Once we have computed the relevant lattice points on the facet $\mathcal{F}$, we are able to compute a solution to the (HMBP) instance. This amounts to solving the following system of linear Diophantine equations:

$$\lambda_1 \boldsymbol{f}_1 + \lambda_2 \boldsymbol{f}_2 + \ldots \lambda_m \boldsymbol{f}_m = \boldsymbol{v} - \boldsymbol{r}, \text{ where } \lambda_i \in \mathbb{Z}_+, \text{for } i \in \{1, \ldots, m\}. \tag{3.52}$$

There are algorithms that solve equations of the above type optimally if an all integral solution exists (FILGUEIRAS & TOMÁS, 1993). As the residual $\boldsymbol{r}$ is not given a priori, the above equation without $\boldsymbol{r}$ subtracted does not necessarily have an integral solution. Therefore, we will develop another polynomial time approach by using a directed acyclic graph (DAG):

Let $D = (V, A)$ be a directed acyclic graph that is implicitly given by the origin and all translated copies of $\mathcal{C}_1$ within the cone $\mathcal{C}_n$. The arcs $a_{11}, \ldots a_{1m}$ are given by the vectors $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m$. They lead to nodes $v_{11}, \ldots, v_{1m}$ within $\mathcal{C}_1$. From each of these nodes, again m arcs given by the vectors $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m$ lead to the (not necessarily distinct) nodes $v_{21}, \ldots, v_{2m}, v_{2m+1}, \ldots, v_{\binom{m+1}{2}}$ within $\mathcal{C}_2$, and so on up to $\mathcal{C}_n$ with nodes $v_{n1}, \ldots, \ldots, v_{\binom{m+1}{n}}$. Every arc in A has unit weight. In particular, D is cycle free. The example in Figure 3.17 shows such a construction of D for the case $m = 3$ and $n = 5$.

Proposition 3.36. *The set of all directed paths in D starting at the origin node form a matroid $\mathcal{M}$. The paths of length n ending in $\mathcal{C}_n$ are the bases of $\mathcal{M}$, and therefore $rank(\mathcal{M}) = n$.*

Proof. D is directed and cycle free. Therefore, the set of all directed paths in D starting at the origin satisfies the matroid property. □

Corollary 3.37. *A solution to* (HMBP) *according to Theorem 3.31 can be computed using a greedy strategy.*

Proof. The computation of a solution to an (HMBP) instance $\mathcal{B}$ then corresponds primarily to finding a path in the graph $D = (V, A)$ of length n starting at the origin, and ending at a node $v_{xy} \in V$ that has minimum distance to $\boldsymbol{v}$. As D is a matroid, applying a greedy strategy will deliver the optimal solution. A solution may be computed by a standard shortest path algorithm as Dijkstra's algorithm. □

The graph $D = (V, A)$ is growing vastly with increasing dimension of the instance $\mathcal{B}$. Thus, applying a standard approach is not very promising. We may solve the problem much more efficiently if we take advantage of a so-called 'go direction' which is given quasi free of charge in the shape of the vector $\boldsymbol{v}$.

We may restrict the consideration of nodes to these nodes located in a corridor around the line interconnecting $\mathbf{0}$ and $\boldsymbol{v}$. As $\mathcal{C}_n$ and $\mathcal{G}$ are completely symmetrical, the facets $\mathcal{F}_2, \ldots, \mathcal{F}_n$, where $\mathcal{F}_i := \text{conv}(i\boldsymbol{f}_1, \ldots, i\boldsymbol{f}_\ell)$ are symmetrically built from elements that are congruent to $\mathcal{F}_1$, and thus have identical diameter. Therefore, we might restrict the search by taking into account only nodes within a radius of less than half the diameter of the facet $\mathcal{F}_1$ without losing paths potentially leading to an optimal solution. This amounts to considering only m nodes in every layer $\ell \in \{1, \ldots, n\}$, namely those defining that simplex on the facet $\mathcal{F}_\ell$ that is intersected by the straight line interconnecting $\mathbf{0}$ and $\boldsymbol{v}$. Any other vertices on $\mathcal{F}_\ell$ would have a greater distance to the intersection point. Moreover,

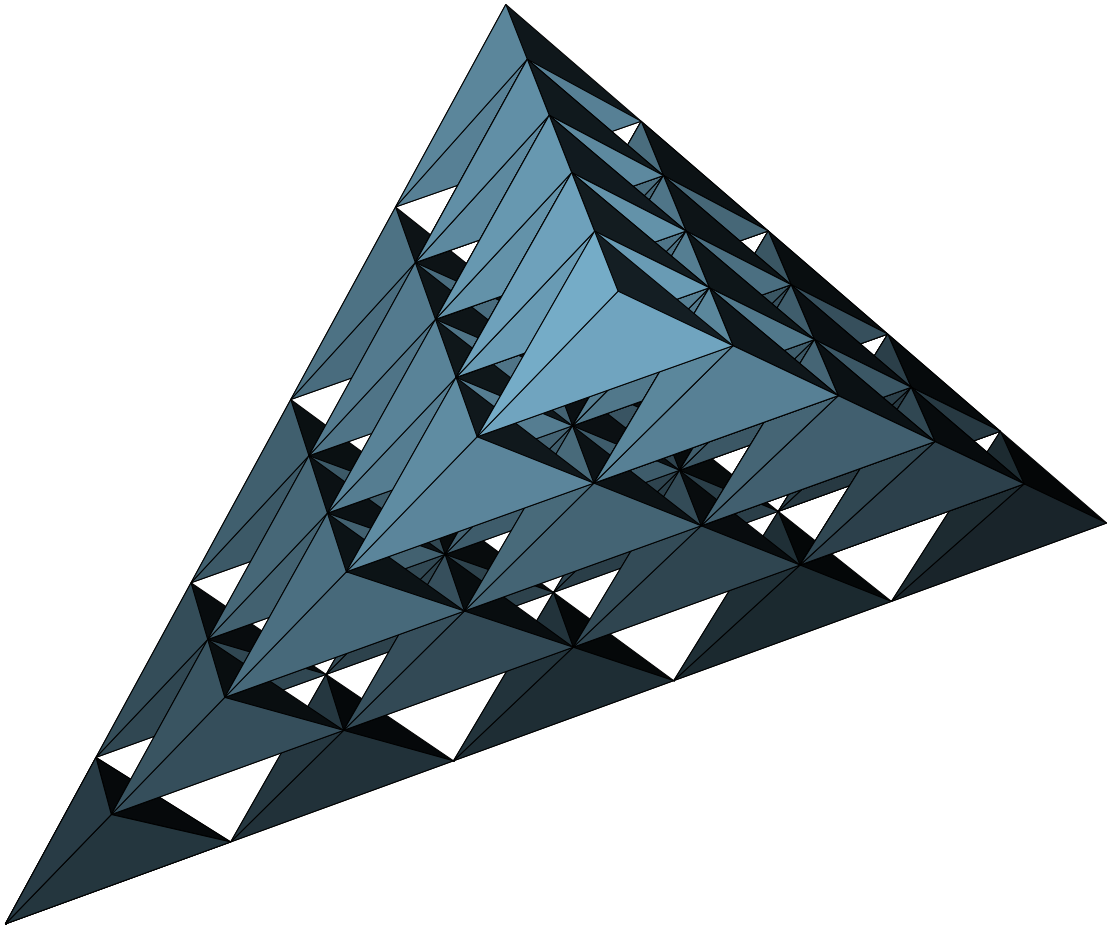


Figure 3.17: Nesting of cones $\mathcal{C}_1, \ldots, \mathcal{C}_5$ for an (HMBP) instance with $m = 3$ and $n = 5$. The cones are generated by translated copies of $\mathcal{C}_1$ that is located in the very lower left corner. Every cone $\mathcal{C}_\ell$ is built from exactly $\sum_{i=1}^{\ell} \frac{n(n+1)}{2}$ translated copies of $\mathcal{C}_1$ (including $\mathcal{C}_1$ itself), and there is a canonical triangulation of each facet $\mathcal{F}_\ell$ using just as many points. These are exactly the combinations of ℓ not necessarily pairwise different points defining $\mathcal{F}_1$.

it suffices to confine ourselves to that vertex in every layer that has minimum distance to the straight line connecting $\mathbf{0}$ and $\boldsymbol{v}$. This is due to the fact that $\mathcal{G}$ is symmetric: every node has exactly m outgoing arcs, and for any path to a node $v_{\ell h_\ell}$ consisting of arc $a_{1k_1}, \ldots, a_{\ell k_\ell}$ it applies that any arbitrary permutation of $\{a_{1k_1}, \ldots, a_{\ell k_\ell}\}$ identifies the same node $v_{\ell h_\ell}$. We summarize the above considerations in the following lemma.

Lemma 3.38. *Given an* (HMBP) *instance $\mathcal{B} = (\boldsymbol{v}, \boldsymbol{w}, C, n)$ and m points defining a facet $\mathcal{F}_1 \subseteq \mathbb{R}^m$ that is intersected by the straight line interconnecting $\mathbf{0}$ and $\boldsymbol{v}$, and $\boldsymbol{v}/n \in \operatorname{conv}(0, \mathcal{F}_1)$. Then there is an $\mathcal{O}(m \cdot n)$ algorithm computing a solution to $\mathcal{B}$ using at most $n + 1$ bins.*

Proof. In each iteration of the shortest path algorithm, no more than m nodes have to be considered. This is repeated exactly n times. □

Note that the graph $D = (V, A)$ needs not to be constructed explicitly for the purpose of computing a solution. In fact, we might solve the problem straightforwardly by a greedy approach that chooses

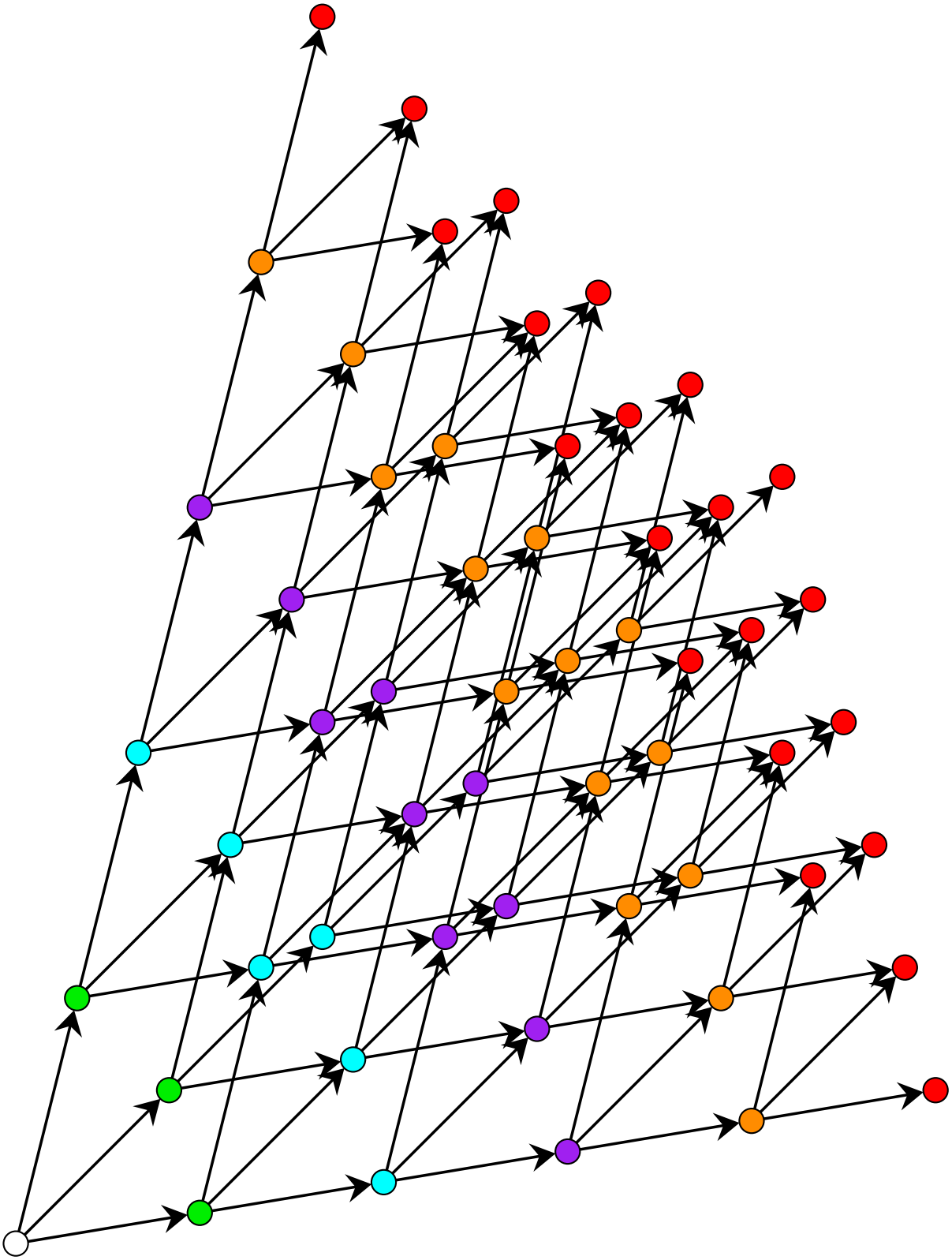


Figure 3.18: Construction of the directed graph $D = (V, A)$ from the (HMBP) instance in Figure 3.17 with $m = 3$ and $n = 5$. All nodes in G correspond to the intersection points of line segments in Figure 3.17. Nodes that belong to the same facet $\mathcal{F}_\ell$ of $\mathcal{C}_\ell$ are colored identically. Clearly, nodes belonging to $\mathcal{F}_\ell$ have distance ℓ from the start node.

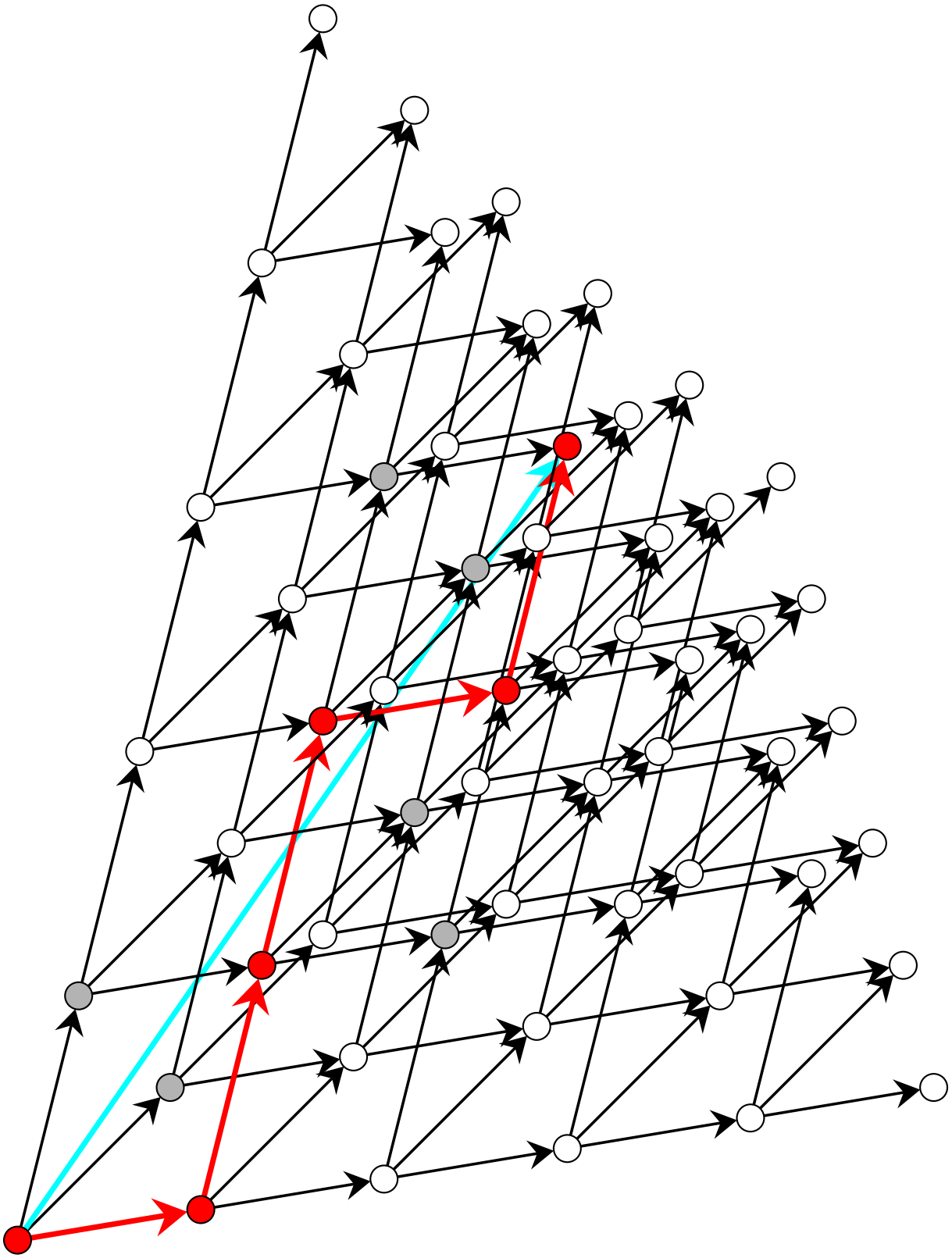


Figure 3.19: Consider the graph D constructed in Figures 3.17 and 3.18. An optimal path (the bold red arcs) in D can be constructed by using a shortest path algorithm and a go–direction (marked by the a long light blue arrow). In every step of the algorithm, only 3 nodes have to be considered (the two grey ones and the eventually chosen red one). Note that the distances in 3–dimensional space have to be considered.

a vector $\boldsymbol{f}_\ell \in \mathcal{F}_1$, $\ell \in \{1, \ldots, m\}$ such that the endpoint of the path generated by the sum of chosen vectors has minimum distance to the straight line interconnecting $\mathbf{0}$ and $\boldsymbol{v}$.

Eventually, we conjecture that an optimal solution may be produced by generating the path exactly the other way round. That is, starting at the vertex $\boldsymbol{v}$, iteratively subtract n vectors from $\boldsymbol{f}_\ell \in \mathcal{F}_1$, $\ell \in \{1, \ldots, m\}$ such that no component becomes less than zero; choose always that vector $\boldsymbol{f}_\ell \in \mathcal{F}_1$, $\ell \in \{1, \ldots, m\}$ that minimizes the distance from the straight line interconnecting $\mathbf{0}$ and $\boldsymbol{v}$. The residual after n subtractions is either less or equal than $\mathbf{0}$ (then there is a solution using n bins), or a feasible bin pattern if possibly occurring negative components are projected to zero. Clearly, as done above, these components might be subtracted from the other patterns used in the solution.

3.5 Summary

In this chapter, we applied ourselves to the HIGH MULTIPLICITY BIN PACKING PROBLEM. We gave an overview about ILP models and solution approaches of the problem. We presented lower and upper bounds on the charge of a bin and determined the notion of irreducibility of instances. Moreover, we illuminated the prerequisites for the feasibility of an instance. We introduced the notion of dominating bin patterns and motivated their importance. We proved an upper bound on the number of dominating bin patterns, and utilized this concept to develop two novel algorithms for the problem: For the 2–dimensional case, we formulated an optimal polynomial algorithm, that extends the approaches presented by AGNETIS & FILIPPI (2005) and MCCORMICK ET AL. (2001). We gave some combinatorial insights into the m–dimensional case and the associated lattice polytopes. Based on these considerations, we developed an efficient algorithm that computes a solution to the problem that utilizes at most one more bin than the optimal solution. It can be shown that – under certain circumstances – an optimal solution is calculated. Furthermore, we have shown that the constructed solutions have a size that is polynomial in the size of the input.

4 Bin Packing Constraints and Concave Constraints

But we all know the world is nonlinear.

— Harold Hotelling to George Dantzig in DANTZIG (2002)

In this chapter, we present a framework to evaluate a global constraint, namely the *Bin Packing Constraints*, jointly with an arbitrary number of instances of a fairly broad, fundamental class of further constraints, which we call *Concave Constraints*. Concave constraints include in particular all linear constraints such as linear inequalities and disequations. Moreover, many logical operations may be expressed by means of Concave Constraints. Therefore, many problems modeled as Linear Programs (LP) or Mixed Integer Linear Programs (MILP) can be easily transformed into our framework. Nevertheless, this conception goes far beyond the scope of Linear Programming: most logical operations might be modeled easily by aid of concave constraints as well as many non-linear constraints in the form of arbitrary concave functions. In the first place, this allows a sophisticated modeling for many applications without the limitations that arise for example from a linearization of nonlinear models.

After a short introduction to Constraint Programming (CP) in Section 4.1, we will introduce the framework of Bin Packing Constraints and concave constraints in Section 4.2. We will give an overview on selected applications that may be modeled in the framework of Bin Packing Constraints and concave constraints in Section 4.3. In Section 4.4, we will give an algorithmic framework to jointly evaluate Bin Packing Constraints and concave constraints.

4.1 Preliminaries

We will give a short introduction to the paradigm of Constraint Programming in the following.

4.1.1 Constraint Programming

Constraint Programming (CP) is a programming paradigm, which states relations between variables in form of constraints. For example, $x < y$ is a simple constraint between two variables, i.e. it is a *binary* constraint. Constraint Programming is a type of *declarative programming*. In contrast to *imperative programming*, where algorithms are explicitly specified in order to deterministically produce a result, declarative programming explicitly specifies the properties of the output, and leaves the algorithmic implementation to a supporting software. Those properties are usually described by means of a *Constraint Programming language* (cf. MILANO (2003)).

Definition 4.1 (Constraint)**.** *Let $X = \{x_1, \ldots, x_n\}$ be a set of variables, and $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ the set of their (finite)* domains*. A* constraint *C on the set X a is a function*

$$C(X) : \mathcal{D} \longrightarrow \{true, false\}. \tag{4.1}$$

A tuple $\boldsymbol{v} \in \mathcal{D}$ for which $C(X)$ delivers "true" is said to satisfy *C. The* solution space $\mathcal{S}_C \subseteq \mathcal{D}$ *is the set of all elements of $\mathcal{D}$ for which $C(X)$ delivers "$true$". $|X|$ is called the* arity *of the constraint $C(X)$.*

Definition 4.2 (Constraint Satisfaction Problem (CSP)**.** *A* Constraint Satisfaction Problem (CSP) *is a triple $(X, \mathcal{D}, \mathcal{C})$. It consists of a set of* variables $X = \{x_1, \ldots, x_n\}$*, a set of (finite)* initial domains $\mathcal{D}_0 = \{D_0(x_1), \ldots, D_0(x_n)\}$ *and a set of* constraints $\mathcal{C} = \{C_1, \ldots, C_m\}$ *between variables from X. For $\ell \in \{1, \ldots, n\}$, $D_0(x_\ell)$ is the initial (finite) set of possible values of variable x_ℓ. A constraint C_ℓ on the ordered set of variables $X(C_\ell) = \{x_{i_1}, \ldots, x_{i_r}\}$ implies a subset $T(C_\ell) \subseteq D_0(x_{i_1}) \times \ldots \times D_0(x_{i_r})$ that contains all combinations of value assignments for the variables $x_{i_1}, \ldots, x_{i_r}$ that satisfy C_ℓ.*

A solution to the Constraint Satisfaction Problem $(X, \mathcal{D}, \mathcal{C})$ is a vector $\hat{\boldsymbol{x}} := (\hat{x}_1, \ldots, \hat{x}_n) \in D_0(x_1) \times \ldots \times D_0(x_n)$, such that for all $\ell \in \{1, \ldots, m\}$, it is $C_\ell(\hat{\boldsymbol{x}}) = true$.

An r-tuple τ of $D_0(x_{i_1}) \times \ldots \times D_0(x_{i_r})$ is called *valid* if $\tau \in D(x_{i_\ell})$, for all $\ell \in \{1, \ldots, r\}$. A constraint C is said to be *consistent* if there exists at least one valid r-tuple τ in $T(C)$. A value $a \in D(x_{i_\ell})$ is *consistent* with a constraint C with $x_{i_\ell} \in X(C)$ if there is an r-tuple $\tau \in T(C)$ such that $\tau_{i_\ell} = a$ and τ is valid. In this case, we say that τ is a *support* of a in C. A constraint C is *arc-consistent* if for each variable $x_{i_\ell} \in X(C)$, and for all $a \in D(x_{i_\ell})$, there exists a support τ for a in C. A Constraint Satisfaction Problem $(X, \mathcal{D}, \mathcal{C})$ is *arc-consistent* if there is no empty domain in $\mathcal{D}$ and all the constraints in $\mathcal{C}$ are arc-consistent.

A general consistency property A, e.g. arc-consistency, can be enforced on a CSP by removing all the unsupported values from the domains of variables. By enforcing A on a CSP $(X, \mathcal{D}, \mathcal{C})$, we mean applying algorithms that yield a new CSP $(X, \tilde{\mathcal{D}}, \mathcal{C})$ with $\tilde{\mathcal{D}} \subseteq \mathcal{D}$ that has the property A, and has the same set of solutions as $(X, \mathcal{D}, \mathcal{C})$. Usually, a constraint is associated with a *filtering algorithm* that evaluates the constraint for value assignments in order to prune the domains of the associated variables (*domain reduction*) and thus to induce a consistency property A. This procedure is called *constraint propagation*. Instead of using linear relaxations for pruning the search tree as it is done in Integer Linear Programming, CP uses a variety of bounding techniques based on constraint propagation in order to prune the domains of variables and to generate new constraints to consequently reduce the search space. Due to the fact that full constraint satisfaction is $\mathcal{NP}$–complete, the implementation of constraint propagation within a solver is usually incomplete. This means that some but not all the consequences of constraints are deduced. In particular, constraint propagation cannot detect all inconsistencies. Consequently, tree search algorithms must be implemented in order to determine consistency (cf. APT (2003); HOOKER (2000)).

4.1.2 Global Constraints

The smallest particles in Constraint Programming are *elementary constraints*. These include for example basic algebraic and logic operations, especially *unary* and *binary constraints* interrelating one variable to a constant or another variable. Unlike these, *global constraints* are complex structures incorporating many variables and relations. They might be described by a variety of elementary constraints. Global constraints represent the powerful tools for modeling problems on a high level. Therefore, they play a fundamental role in Constraint Programming. A very common representative of global constrains is the constraint *all-different*$(x_{i_1}, \ldots, x_{i_r})$ which enforces all of its associated variables $x_{i_1}, \ldots, x_{i_r}$ to assume pairwise different values in a possible solution. Global constraints often implement whole combinatorial optimization problems such as the Traveling Salesman Problem (TSP) (APPLEGATE ET AL., 2007; REINELT, 1994). For this reason, their propagation is often $\mathcal{NP}$–complete.

A large number of global constraints have been proposed over the years (APT, 2003; BELDICEANU & CONTJEAN, 1994; FOCACCI ET AL., 2002; HOOKER, 2000; REGIN & RUEHER, 2000). See BELDICEANU

ET AL. (2007) for a unifying overview, or SIMONIS (2007) for selected application models using global constraints. However, there are only few results on the *algorithmic evaluation* of global constraints since such an evaluation often amounts to solving an $\mathcal{NP}$–hard problem. See GRANDONI & ITALIANO (2006); MEHLHORN & THIEL (2000) for examples of algorithmic results on specific global constraints, and BELDICEANU (2000); BELDICEANU ET AL. (2004) for an attempt to quite a generic approach. For the knapsack constraint, approximation algorithms are used to gain consistency in SELLMANN (2003). The combination of global constraints with other elementary constraints has been considered from a theoretical view in MAHER (2002). To our knowledge, the algorithmic evaluation of these combinations has not systematically been taken into account so far. This is unfortunate since this is of particular practical interest: Jointly evaluating a set of constraints may help reduce the search space much more aggressively than evaluating constraints quite independently of each other.

However, in the literature, constraint propagation is often interpreted as merely pruning the ranges of individual variables (domain reduction). Driven by our research, we tend to a much more general interpretation: Constraint propagation means to us to efficiently manage information about the set of (possibly) feasible solutions such that, at each node of the search tree, it is detected with a very high probability, whether the subtree rooted at this tree node does not contain any feasible solutions. Clearly, the reduction of individual variables' domains is a special case of that: whenever the range of a variable becomes empty during the search, it is detected that the subtree rooted at this tree node does not contain any feasible solutions.

4.2 Bin Packing Constraints and Concave Constraints

4.2.1 Background

We consider a certain global constraint, the *Bin Packing Constraint*, which turns out to be a natural common abstraction for specific aspects of a wide variety of applications. Among many other novel global constraints, the Bin Packing Constraint has been firstly mentioned as an item in the systematic presentation of global constraints by BELDICEANU (2000). Our application-oriented research led us to a slightly more general definition. We observed that various aspects of many applications can be particularly well modeled by a combination of one or more instances of the Bin Packing Constraint and constraints of a fairly general class, which we call *Concave Constraints*. This latter type of constraints includes in particular all linear constraints such as linear inequalities and disequations, and some fundamental classes of logic operation such as implications and disjunctions. In fact, it also encloses many non-linear constraints. Therefore, the class of problems modeled by concave constraints goes far beyond the class of problems which can be modeled as Mixed Integer Linear Programs (MILP).

One of the fundamental concepts of constraint programming is the recursive partition of the search space into smaller and smaller subsets as a consequence of descending from node to node in the search tree. More formally, each tree node v is associated with a subset S_v of the search space. For an arc (v, w) of the search tree we have $S_w \subseteq S_v$. Typically, S_v is split into two or more (usually disjoint) partition sets by splitting the ranges of one or more variables into (disjoint) subranges. Each partition set then constitutes a tree arc emanating from v. Splitting the range of a variable might be achieved by introducing additional linear inequalities. In Section 4.2.2, we will see that this fits well with our technique.

However, prior to descending from v to w, first an attempt is made to construct a certificate that the subtree rooted at w does not contain any feasible solution. Basically, this means calling an algorithm that delivers "infeasible" or "possibly feasible", and "infeasible" only if this subtree indeed does not

contain any feasible solution. Clearly, the quality of the algorithm is evaluated against two criteria: its run time and its accuracy, that is, how often it delivers "infeasible" for infeasible subtrees.

4.2.2 Concave Constraints

In this work, a constraint C is defined on some specific space $\mathbb{R}^\mu$. The dimensions may have a structure, as in our case we may speak of $\mathbb{R}^{m\times n}$ instead of $\mathbb{R}^\mu$. Therefore, we tend to a slightly different definition of the term "constraint" from that in Section 4.1.1.

Definition 4.3 (Constraint)**.** *A* constraint C *associated with* $\mathbb{R}^\mu$ *is then a function*

$$C : \mathbb{R}^\mu \longrightarrow \{true, false\}\,. \tag{4.2}$$

The solution space $S_C \subseteq \mathbb{R}^\mu$ *is the set of all elements of* $\mathbb{R}^\mu$ *for which* C *delivers "*$true$*".*

Definition 4.4 (Concave Constraint)**.** *A constraint* C *on the space* $\mathbb{R}^\mu$ *is called* concave *if the complement* $\mathbb{R}^\mu \setminus S_C$ *of the solution space* S_C *is convex.*

It is easy to see that the following fundamental classes of basic constraints are concave:

- *Linear inequalities and disequations:* there are $r_1, \ldots, r_\mu$, $s \in \mathbb{R}$ and $\odot \in \{\leq, \geq, <, >, \neq\}$ such that the constraint evaluates to *true* for $\boldsymbol{x} \in \mathbb{R}^\mu$ if, and only if

$$\sum_{i=1}^{\mu} r_i \cdot x_i \;\odot\; s\,. \tag{4.3}$$

  Clearly, this implies that a linear equation may be expressed by concave conditions, too, namely by two linear inequalities.

- *Logic implications* of the form $z_1 \odot_1 c_1 \Rightarrow z_2 \odot_2 c_2$, where z_1 and z_2 are real–valued variables, $\odot_1, \odot_2 \in \{\leq, \geq, <, >\}$, and $c_1, c_2 \in \mathbb{R}$.

- *Binary disjunctions* of the form $z_1 \odot_1 c_1 \vee z_2 \odot_2 c_2$, where z_1 and z_2 are real–valued variables, $\odot_1, \odot_2 \in \{\leq, \geq, <, >\}$, and $c_1, c_2 \in \mathbb{R}$. We can also model XOR-conditions, as each XOR-condition can be expressed by two OR-conditions.

- *Constraints using arbitrary concave or convex functions* of the form $g(x_1) \odot x_2$, where $\odot \in \{\leq, \geq\}$, and $g(x)$ is an arbitrary concave or convex function. Clearly, when using a concave function we have to choose $'\leq'$ for $\odot$, and $'\geq'$ in case of a convex function.

- *Constraints involving arbitrary convex sets* of the general form $\boldsymbol{x} \notin \mathrm{conv}(\boldsymbol{f}_1, \ldots, \boldsymbol{f}_\ell)$. These might be for example geometric constraints such as $x \notin \mathrm{circle}(y, 2)$, which means x is not allowed to be within the circle around y with radius 2.

Proposition 4.5. *All constraints from Mixed Integer Linear Programming (MILP) can be modeled by means of concave constraints. These are in particular linear inequalities, integrality constraints, Big-M constraints, and Special Ordered Sets (SOS).*

Note that modeling integrality constraints by concave constraints might require a very large number of disjunctive constraints. Therefore, it makes more sense to impose integrality by discrete domains.

There are two common techniques for descending in a search tree: one is to add bounds on the objective function as it is realized in Branch&Bound algorithms. The other one is to add cutting planes that narrow the solution space as it is realized in Branch&Cut algorithms. Both approaches amount to adding linear inequalities. These are obviously concave. Therefore, these techniques are compatible with our framework.

4.2.3 The Bin Packing Constraint

Definition 4.6. *Let $\mathbb{I} := \{1, \ldots, m\}$ be a set of items of sizes $a_1, \ldots, a_m$. Furthermore, let $\mathbb{B} := \{1, \ldots, n\}$ be a set of bins of capacities $c_1, \ldots, c_n$. The* Bin Packing Constraint

$$\text{bin–packing}((\pi_1, \ldots, \pi_m), (a_1, \ldots, a_m), (c_1, \ldots, c_n)) \tag{4.4}$$

evaluates to true if, and only if there exists a distribution of items to bins such that all items are distributed to bins, while no bin capacity must be exceeded. Formally, there must be an assignment function from items to bins

$$\pi : \mathbb{I} \longrightarrow \mathbb{B}, \text{ such that } \sum_{i=1}^{m} \left\{ a_i \,\middle|\, \pi(i) = j \right\} \leq c_j, \text{ for all } j \in \{1, \ldots, n\}. \tag{4.5}$$

For notational convenience in (4.4), we set $\pi_i := \pi(i)$.

The constraint evaluates to "true", if and only if such an assignment exists.

Equivalently, we may require the existence of binary variables x_{ij} for all $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$ such that

$$x_{ij} := \begin{cases} 1, \ \pi(i) = j, \\ 0, \ \textit{otherwise}. \end{cases} \tag{4.6}$$

Then, the signature of the constraint is

$$\text{bin–packing}((x_{11}, \ldots, x_{mn}), (a_1, \ldots, a_m), (c_1, \ldots, c_n)). \tag{4.7}$$

The constraint evaluates to "true" if, and only if

$$\sum_{i=1}^{m} a_i \cdot x_{ij} \leq c_j, \quad \textit{for all } j \in \{1, \ldots, n\}, \textit{ and} \tag{4.8}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \textit{for all } i \in \{1, \ldots, m\} \tag{4.9}$$

For the formulation of concave constraints in general, the formulations (4.8) and (4.9) in Definition 4.6 that use binary assignment variables are often more obvious. Therefore, we will use this formulation in the following.

Proposition 4.7. *Any Bin Packing Constraint formulated in π-notation (4.4) can be equivalently transformed into a Bin Packing Constraint using binary x_{ij}-variables (4.7).*

In general, the decision whether such a distribution exists is $\mathcal{NP}$–hard in the strong sense (GAREY & JOHNSON, 1979). However, the challenge is weaker regarding constraint propagation. Here, we are satisfied by an algorithm that delivers "infeasible" or "possibly feasible" such that "infeasible" is only delivered for instances that are definitely infeasible (cf. HOOKER (2000); FOCACCI ET AL. (2002)). In Section 4.4 we develop a filtering algorithm that uses a multi–stage relaxation. In a similar way as linear relaxations are used in Branch&Bound algorithms to compute bounds and cut off infeasible subtrees at nodes not satisfying the bounds, we determine a certificate of infeasibility by considering the vertices of a linear relaxation of the solution space together with concave constraints.

4.2.4 Concave Constraints in the Frame of the Bin Packing Constraint

We will discuss a variety of concave constraints that may occur in the frame set by Section 4.2.3. This might cover most highly generic constraints that are time and again encountered in practical applications such as those discussed in Section 4.3.

Unary constraints

Assignment restrictions form very common additional unary constraints. There, an item $i \in \{1, \ldots, m\}$ can only be assigned to a certain bin from an item–specific selection $S_i \subseteq \{1, \ldots, n\}$. This can be expressed by linear disequations: $\pi(i) \neq j$, for all $j \in \{1, \ldots, n\} \setminus S_i$. This immediately translates to a formulation using binary variables: $x_{ij} = 0$, for all $j \in \{1, \ldots, n\} \setminus S_i$. As mentioned in Section 4.2.2, disequations are concave.

If S_i is an interval, $S_i = \{a, \ldots, b\} \subseteq \{1, \ldots, n\}$ $(1 \leq a \leq b \leq n)$, the restriction to S_i can clearly be expressed in a more compact fashion by concave constraints: $\pi(i) \geq a$ and $\pi(i) \leq b$. On the other hand, consider the case that the set of feasible bins for an item i consists of exactly two intervals, $\{a_1, \ldots, b_1\}$ and $\{a_2, \ldots, b_2\}$ $(1 \leq a_1 \leq b_1 < a_2 \leq b_2 \leq n)$. Then we can simply introduce analogous linear inequalities, $\pi(i) \geq a_1$ and $\pi(i) \leq b_2$, and add the concave condition $\pi(i) \leq b_1 \vee \pi(i) \geq a_2$. Conditions of that type might be translated to the formulation using binary variables either by adding separate unary constraints for each bin, or, more elegantly, by means of a set of linear inequalities, e.g.

$$\sum_{i=1}^{a-1} x_{ij} = 0 \text{ and } \sum_{i=b+1}^{n} x_{ij} = 0. \tag{4.10}$$

Binary constraints

Identity constraints and *exclusion constraints* are important examples of binary constraints. For example, for $i_1, i_2 \in \{1, \ldots, m\}$ we have the strict identity $\pi(i_1) = \pi(i_2)$, the loose identity $\pi(i_1) \geq \pi(i_2) - h_1$ and $\pi(i_1) \leq \pi(i_2) + h_2$, and exclusions of the form $\pi(i_1) \neq \pi(i_2)$ or, more generally, $\pi(i_1) = h_1 \Rightarrow \pi(i_2) \neq h_2$. As discussed in Section 4.2.2, conditions of all of these types are concave or may be modeled by sets of concave conditions.

Precedence constraints are another important type of binary constraints. Precedence means there is a pair of items $i_1, i_2 \in \{1, \ldots, m\}$ such that i_1 must be assigned to some bin which is a predecessor of that bin item i_2 is assigned to. This can be obviously expressed by a linear inequality, $\pi(i_1) \leq \pi(i_2)$, which is again a concave constraint. In scheduling applications, it is common to connect two precedence constraints by an XOR operation to express the situation 'either A before B, or B before A'. Clearly, this kind of constraint is nonlinear.

Any of the above constraints might be easily translated into the formulation using binary variables by means of implications.

Constraints of higher arities

Sometimes the number of items per bin is restricted regardless of whether the capacity of the bin would suffice for further items. Formally, there is a positive integral number d_j for each $j \in \{1, \ldots, n\}$

such that at most d_j items may be put in the j–th bin. This kind of constraint amounts to a linear inequality and is thus concave:

$$\sum_{i=1}^{m} x_{ij} \leq d_j \,. \tag{4.11}$$

This can be generalized in two ways: first, the number of items is not restricted for a single bin, but for a set of bins, and second, only the number of items from a subset of $\{1, \ldots, m\}$ is restricted. In the common generalization of both options, there are non–empty subsets $I_1, \ldots, I_k \subseteq \{1, \ldots, m\}$ of items, non–empty subsets $B_1, \ldots, B_k \subseteq \{1, \ldots, n\}$, and a positive integral value e_ℓ for each $\ell \in \{1, \ldots, k\}$. For $\ell \in \{1, \ldots, k\}$, the ℓ–th condition then reads

$$\sum_{i \in I_\ell,\, j \in B_\ell} x_{ij} \leq e_\ell \,, \tag{4.12}$$

which again is a concave constraint.

Nonlinear Concave Constraints

Many applications from computational geometry, engineering sciences, and financial optimization comprise nonlinear functions. There are numerous mathematical functions that are either concave or convex . Given a concave function $f(x)$, the constraint $f(x) \leq y$ is clearly concave. Given a convex function $g(x)$, the constraint $-g(x) \leq y$ or $g(x) \geq y$ is concave as well. Therefore, a variety of basic concave or convex functions such as $x^2, \sqrt{x}, e^x, \log x, 1/x$ can be used within the framework of concave constraints. Periodic functions such as trigonometric functions and arbitrary curves that are neither convex nor concave can be at least used piecewise on a specific interval where they have a definite convexity or concavity property.

Various optimization problems involve nonlinear objective functions. In this case, they can usually be linearized by minimizing or maximizing an auxiliary variable and formulating a corresponding side constraint that contains the nonlinear function.

4.3 Selected Applications from Operations Research

In this section, we show that combinations of the Bin Packing Constraint and concave constraints arise quite naturally in a broad domain of problems from Operations Research and Combinatorial Optimization and, therefore, are of particular practical interest. We will focus on a selection of $\mathcal{NP}$–hard problems in order to demonstrate that, and in which way, a combination of the Bin Packing Constraint and concave constraints may be applied to them. Many of the problems described here can be found in the literature in some form or another (cf. BRUCKER (2007); GAREY & JOHNSON (1979); KELLERER ET AL. (2004); NEMHAUSER & WOLSEY (1988)).

Note that the application–oriented discussion in this section does not include any algorithmic considerations or any kind of quantitative analysis, it is a purely qualitative analysis of potential applicability. Recently, SIMONIS (2007) presented some industrial applications that might be modeled by means of global constraints in general. To our knowledge, the applicability of a specific set of constraints in view of a variety of problems is not systematically addressed in the literature. However, we strongly believe that studies of this kind may contribute useful additional insights in their own right, which complement algorithmic considerations.

We will only consider $\mathcal{NP}$–hard algorithmic problems because constraint programming might be useful for this kind of algorithmic problem in the first place. We will consider both, *feasibility problems*

and *optimization problems*. Our presented selection of applications comprehends a choice of well-known problems as well as some less obvious applications of the Bin Packing Constraint.

Like any other constraint, the Bin Packing Constraint is a matter of *feasibility* in the first place. However, we argue that, within an enumerative frame such as *Branch&Bound* or *Constraint Programming*, the Bin Packing Constraint is particularly useful for the following two generic optimization problems:

- The first type of optimization problem results from the Bin Packing Constraint when we leave the number of bins open. In other words, the objective is to minimize the number of bins to be used. Typically, but not necessarily, the capacities of all bins are then identical.
- The second type of optimization problem is a kind of min–max problem. Here, the number of bins is fixed, but no bin capacities are given. Instead, there is a factor $f_j \in \mathbb{R}^+$ for each $j \in \{1, \ldots, n\}$. The problem is to distribute the items over the bins such that the maximal f–weighted workload is minimized, which is

$$\min \left\{ \max_j \left\{ f_j \cdot \sum_{i=1}^{m} \{a_i \,|\, \pi(i) = j\} \right\} \right\}. \tag{4.13}$$

In the following sections, we present a variety of problems of the above types. For each problem, we give a problem definition and a modeling option in terms of the Bin Packing Constraint and Concave Constraints. Moreover, we will give examples for additional (concave) constraints. Finally, we give information about what is happening during the traversal of the search tree, and evidence for a small number of bins, which is important for an efficient evaluation of the framework.
In many applications, we put special emphasis on the nonlinear constraints. Sections 4.3.9 and 4.3.10 are dedicated to two applications that naturally involve nonlinear objective functions and nonlinear side constraints, respectively.

4.3.1 BIN PACKING and KNAPSACK type problems

We first mention BIN PACKING and MULTIPLE KNAPSACK problems as they yield the most obvious applications of the Bin Packing Constraint. Note that there is a specific global constraint for knapsack problems, which is analyzed in FAHLE & SELLMANN (2002); SELLMANN (2003); TRICK (2001). For an extensive synopsis of different types of knapsack problems and additional constraints see KELLERER ET AL. (2004).

BIN PACKING

A set $\{1, \ldots, m\}$ of items is to be distributed over a certain set of bins. If both the number of bins and their capacities are given, this amounts to a feasibility problem that exactly complies with the Bin Packing Constraint. The natural optimization versions are exactly the two optimization problems stated at the beginning of Section 4.3.

MULTIPLE KNAPSACK

Again, we want to distribute m items to n bins (here: knapsacks). In this variant, not necessarily all items have to be distributed. This is usually due to capacity restrictions. Additionally, the items have a value w_i. The goal is to fill the bins in such a way that the sum of values of packed items is maximal.

As not all items are to be packaged, we introduce a dummy bin $n+1$, which absorbs all items that are not packed into one of the original knapsacks $\{1,\ldots,n\}$. The capacity c_{n+1} of the dummy bin may be set to the sum of all item sizes. Then, we can state the Multiple Knapsack problem as

$$\textit{maximize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}w_i \tag{4.14}$$

$$\text{s.t.} \quad \textit{bin–packing}\,((x_{11},\ldots,x_{m,n+1}),(a_1,\ldots,a_m),(c_1,\ldots,c_{n+1}))$$

$$\text{with } c_{n+1} := \sum_{i=1}^{m} a_i. \tag{4.15}$$

Additional constraints

Generally, arbitrary additional concave constraints are imaginable. Often, there are constraints on the item–to–bin assignments, e.g. exclusion constraints, sequence constraints, or precedence constraints. Exclusion constraints exclude a set S_j of specific items from a bin j. Sequence constraints enforce that all items of a set S_i are packed into pairwise different bins. Precedence constraints ensure an item i is packed before an item j is chosen. An additional capacity criterion might be used in order to distribute values or weights as equal as possible over the bins. This is useful in load balancing problems. For example, then item sizes represent job lengths, and the sum of item sizes assigned to a bin j its makespan.

4.3.2 Set Cover, Set Partitioning, Hitting Set, and Vertex Cover

These and related problems are classical questions in computer science and complexity theory (cf. Garey & Johnson (1979); Karp (1972)). Many $\mathcal{NP}$–hard problems from Operations Research are reducible from one of these problems.

A *vertex cover* for an undirected graph $G = (V, E)$ is a subset $\mathcal{S}$ of its vertices such that each edge has at least one endpoint in $\mathcal{S}$. Vertex Cover denotes the decision problem whether there is a vertex cover of size ℓ for a given graph G. The problem might be modeled by Bin Packing Constraints, by identifying edges with items and nodes with bins. All edges must be covered. This amounts to all items being packed. Usually, we want to know whether a vertex cover of size ℓ exists. Alternatively, we aim at a minimization of the number of vertices (here:bins). As additional constraints, we have assignment restrictions which are naturally given by the underlying graph.

The modeling of Hitting Set is a little bit more sophisticated. Informally, Hitting Set is defined as follows: we are given a collection of subsets $\mathcal{S}$ of a universe $\mathcal{T}$. We are looking for a subset $\mathcal{H} \subseteq \mathcal{T}$ with $|\mathcal{H}| \leq C$ that intersects ('hits') every set in $\mathcal{S}$. We consider one bin of capacity C that is intended to contain exactly the hitting set. There are m items, and n sets of items. The Boolean predicate b_{ij} determines whether item i is part of set j. We use $0/1$–variables y_ℓ that determine whether an item ℓ is chosen in the hitting set. Thus, the capacity constraint is imposed on the sum of y_ℓ for all $\ell \in \{0,\ldots,m\}$. Furthermore, we use assignment variables $x_i \in \{0,1\}$ that determine whether item i in set j is used in the hitting set, i.e. is packed into the bin. Now, we have to ensure that, in every set, at least one item is chosen in the hitting set, i.e. $\sum_{i=1}^{m} x_{ij} > 0$, for all $j \in \{1,\ldots,n\}$. This might be transformed into the regular constraints $z_j = 1$ using auxiliary variables z_j and constraints $x_{ij} > 0 \;\Rightarrow\; z_j = 1$ and $z_j = 1 \;\Rightarrow\; x_{ij} > 0$. Additionally, we need constraints $x_{ij} \leq b_{ij}$ to ensure that items i only represent sets j they are actually contained in. Furthermore, we need constraints

$x_{ij} = 1 \Rightarrow y_i = 1$, for all $i \in \{1, \ldots, m\}$, that ensure every item representing a set is counted in the objective function.

By SET PARTITIONING, we denote the problem to partition a set of elements $\{1, \ldots, m\}$, each of a distinct weight w_ℓ, into k sets such that the weights are distributed to the sets as equally as possible. Thus, the problem is of the min–max type mentioned at the beginning of Section 4.3: we choose k bins and minimize the maximum load under the restriction that every element has been packed.

SET COVER is a classical problem from complexity theory: given a universe $\mathcal{U}$ and a family $\mathcal{S}$ of subsets of $\mathcal{U}$, a *set cover* is a subfamily $\mathcal{C} \subseteq \mathcal{S}$ of sets whose union is $\mathcal{U}$. SET COVER can be modeled in a straightforward way: the set elements are the items, and every set represents a potential bin allocation. Now, we minimize the number of bins subject to the fact that all items are packed, that is all elements are covered. Additionally, we impose conditions that constrain the potential bin allocations to the composition of sets in the instance in order to prohibit arbitrary allocations.

Additional constraints

Besides the problem specific constraints that extend the BIN PACKING formulation, there might be any kind of additional constraints. As the above mentioned problems are fairly general, we desist from a listing of constraints here.

4.3.3 Project Scheduling

In operations research, a project or process is typically decomposed into a set of (atomic) jobs, which we take as the items $\{1, \ldots, m\}$. Each job $i \in \{1, \ldots, m\}$ has a certain duration d_i. There are usually resource restrictions, which limit the possibilities for processing jobs simultaneously (that is, at overlapping processing periods). For example, some jobs may require a certain machine or specifically skilled staff, and only a certain number of them is available. Likewise, the total number of staff may limit the number of jobs to be processed simultaneously.

Our idea for applying the Bin Packing Constraint to Project Scheduling Problems requires a *planning horizon*. On the other hand, if no planning horizon is given (as is often the case in project scheduling), any upper bound on the optimal total duration will do (for example, the total duration of some feasible solution). Now, we partition the planning horizon into intervals, which form the bins. In other words, assigning a job to the j–th bin means processing this task in the j–th interval.

More specifically, the Bin Packing Constraint is to be applied several times simultaneously, once for each type of resource on which tasks may depend (or a selection of resource types). The capacity of such a bin is then the product of the length and the maximal number of jobs that rely on this resource and may be processed simultaneously.[1] The size of an item is the product of the duration of this task and the amount of this resource consumed by the task during its processing.

Clearly, it may happen that a task is processed at a period of time that intersects with more than one of these intervals. Typically, the processing times of the tasks might be small enough such that the error caused by assigning each task entirely to a single interval is negligible. On the other hand, if there are large tasks for which this error is *not* negligible, the following strategy might solve the problem in our framework: Consider a task T with an exorbitant length L. We partition this task into k smaller tasks, $T_1, \ldots, T_k$, and introduce a couple of (concave) constraints to ensure that these jobs are placed in bins as if they still were one large job. More specifically, let d_i denote the duration of task T_i, $i \in \{1, \ldots, k\}$ (in particular, it is $d_1 + \cdots + d_k = L$). For ease of exposition, let all intervals

[1] If the capacity of a resource is not constant throughout the planning horizon or if the demand of a job is not constant throughout its processing time, more complex, yet obvious schemes apply.

have the same length C and this job to be the only one to be considered. Finally, let $\pi(i)$ denote the interval to which T_i is assigned, and let $x_{ij} \in \{0, 1\}$ be defined by the rule: $x_{ij} = 1$ if, and only if, $\pi(i) = j$. Then the following concave conditions ensure that the T_i's act like the original, large job:

$$\pi(i) \leq \pi(i+1) \text{ for all } i \in \{1, \ldots, k-1\}, \tag{4.16}$$

$$\sum_{i=1}^{k} \left\{ d_i \,\middle|\, x_{ij} = 1 \right\} \leq C, \tag{4.17}$$

$$\pi(k) - \pi(1) \leq \lceil L/C \rceil. \tag{4.18}$$

Scheduling problems often appear as pure feasibility problems, that is, all jobs have to be processed within the given planning horizon. However, in other frequent cases, the problem is to schedule all jobs of a project (subject to the various side constraints) such that the *makespan* of the project is minimized. Therefore, this is an example of the type of min–max problem addressed

Additional constraints

Typically, there are precedence or order constraints among the operations. For two operations, i_1, $i_2 \in \{1, \ldots, m\}$, and a real–valued number x, a precedence constraint defined by (i_1, i_2, x) means that at least x units of time must elapse from the completion of i_1 to the start of i_2 (x may be negative). Such a constraint can be translated into a constraint on the intervals of the planning horizon: for j_1, $j_2 \in \{1, \ldots, n\}$, if the end of interval no. j_1 is more than x units before the start of interval no. j_2, then a schedule in which i_1 is in j_2 and i_2 is in j_1 is not feasible. Clearly, this is a concave implication of the type discussed in Section 4.2.4 which is nonlinear.

During the search

It is quite common to descend from one tree node to another one by restricting the start time of a job to given intervals within a planning horizon. This immediately translates into restricting these jobs to subsets of the bins, which is an example of unary constraints as discussed in Section 4.2.4. On the other hand, whenever the resource constraints are very tight, it is also promising to alternatively introduce additional precedence constraints in order to narrow the search space. As we have seen above, additional precedence constraints are no problem either.

Small number of bins

We have full control over the number of bins, because we define the number of intervals to partition the planning horizon freely ourselves.

4.3.4 Job Shop Scheduling and Production Planning

Job shop scheduling problems ask the question how jobs may be scheduled to machines in an optimal way with respect to certain criteria. Usually, we consider an objective such as the minimization of the maximum makespan or the minimization of the total tardiness, i.e. the sum of delays. For an overview of variants of these problems see BRUCKER (2007).

We are given a set $\mathcal{J} = \{1, \ldots, m\}$ of jobs and a set $\mathcal{M} = \{1, \ldots, n\}$ of resources (e.g. machines or employees). Each job $J \in \mathcal{J}$ is composed of a set of tasks $\mathcal{S}_J = \{s_{J1}, \ldots, s_{Jm_J}\}$, which are to

be processed in an ordered sequence $s_{J1} \rightarrow s_{J1} \rightarrow \ldots \rightarrow s_{Jm_J}$. Each task i can be processed by a subset $\mathcal{F}_i \subset \mathcal{M}$ of the resources. Furthermore, each task i has a specific duration d_i. There are usually resource restrictions, which limit the total use of a resource. We define a bin $j \in \mathcal{M}$ per resource with resource restriction c_j. Binary variables x_{ij} denote whether task i is scheduled on resource j or not. A job J has a release time R_J. Let t_i denote the time at which task i starts, and $\mathcal{T} = \bigcup \mathcal{S}_J$ the set of all tasks. We can then state the problem by the following model using concave constraints.

$$\text{minimize } \max_{J \in \mathcal{J}} \left\{ C_J \right\} \tag{4.19}$$

subject to

$$C_i \geq t_{s_{Jm_J}} + d_{s_{Jm_J}}, \qquad \forall\, J \in \mathcal{J}, \tag{4.20}$$

$$t_{s_{J1}} \geq R_J, \qquad \forall\, J \in \mathcal{J}, \tag{4.21}$$

$$t_{s_{i,k+1}} \geq t_{s_{ik}} + d_{s_{ik}}, \qquad \forall\, i \in \mathcal{J},\, 0 < k < m_J, \tag{4.22}$$

$$x_{kj} + x_{lj} > 1 \Rightarrow$$

$$t_{kj} \geq t_{lj} + d_{lj} \;\vee\; t_{lj} \geq t_{kj} + d_{kj}, \quad \forall\, j \in \mathcal{M}, \forall\, k,l \in \mathcal{T} \tag{4.23}$$

$$\textit{bin–packing}\,((x_{11},..,x_{T_{Im_I},j}),(d_1,..,d_{T_{Im_I}}),(c_1,..,c_n)), \tag{4.24}$$

$$x_{ij} \in \{0,1\}, \qquad \forall\, i \in \mathcal{T}, \forall j \in \mathcal{M} \setminus \mathcal{F}_i. \tag{4.25}$$

In the above model, the minimization of the makespan is our objective (4.19). Inequality (4.20) defines the completion time C_J of each job J, that is its starting time plus its duration. Inequality (4.21) states any job cannot start before it is released. For any two consecutive tasks within a job. Inequality (4.22) ensures that the tasks does not start before the previous one was finished. The implications in (4.23) state that two tasks k, l that are scheduled on the same machine do not overlap. The Bin Packing Constraint (4.24) determines that all jobs are scheduled and the capacity restrictions for each resource hold. Finally, (4.25) ensures that a machine is able to handle a specific task.

Additional constraints

In production planning problems, the tasks of a job do not necessarily have to be processed in a given sequence. Therefore, we can drop equation (4.22). Instead, there may be precedence constraints between pairs of tasks. Furthermore, if a planning horizon is given, the question is how to schedule all tasks within this planning horizon. The objective (4.19) is then dropped, too, and one considers a feasibility problem instead. In this case, it makes sense to introduce one Bin Packing Constraint for each machine. The bins are then intervals of the planning horizon as in project scheduling.

During the search

It is common to descend from one tree node to another one by fixing jobs to resources or to subsets of resources.

Small number of bins

We can get approximate solutions by restricting the number of bins to the number of different types of machines by merging k machines of the same type to a single machine with resource capacity k-times the original one. Clearly, the mergence of machines amounts to a relaxation of the original problem.

4.3.5 Load Balancing

This type of optimization problem is closely related to the Bin Packing Constraint. Here the items are *tasks* and the bins are *machines*, for example, the machines in an assembly line. The problem is to distribute the tasks over the machines. For $i \in \{1, \ldots, m\}$, the i–th task requires time a_i. The goal is to minimize the total time of an assembly line, that is, the maximal time required by any machine to process the tasks assigned to this machine. The formal objective to be minimized is then (4.13) with coefficients $f_j = 1$ for all j. The required time of a machine in this application corresponds to the capacity consumption of the items assigned to this machine, which is the sum of the items' sizes. This is exactly the type of min–max problem addressed at the beginning of Section 4.3.

This simple model does not allow the incorporation of *reset times*. A reset time might be necessary between two jobs to recalibrate the machine or in order to change parts of its setup. The major problem here is due to the fact that the reset times may be different for each pair of jobs: this means that the temporal order of the items on a machine matters. For example, if two items of different types have to be processed immediately after each other on the same machine, it may be necessary to recalibrate that machine in the meantime. In many cases, the reset times are too large to be ignored. If the reset times are more or less identical, they can be simply added to the durations of the tasks. Otherwise, we need a completely different model like the following: The items to be distributed are *pairs* of tasks, and the size of such a pair is the duration of the first item of this pair plus the reset time. The challenge is then to assign a set of these pairs to the individual machines. Clearly, this approach requires additional constraints to ensure that these pairs form an appropriate chain of tasks.

So for a pair $i_1, , i_2 \in \{1, \ldots, m\}$ of tasks and for $j \in \{1, \ldots, n\}$, let $x_{i_1 i_2 j} \in \{0, 1\}$ denote whether the pair (i_1, i_2) is assigned to machine j. Thus, we impose the concave restriction

$$\sum_{j=1}^{n} x_{i_1 i_2 j} \leq 1. \tag{4.26}$$

To enforce a chain on each machine, we introduce the following four concave constraints for each pair (i_1, i_2) and each machine $j \in \{1, \ldots, n\}$:

$$x_{i_1 i_2 j} = 1 \Rightarrow \sum_{i=1}^{m} x_{i i_1 j} \leq 1, \quad x_{i_1 i_2 j} = 1 \Rightarrow \sum_{i=1}^{m} x_{i i_1 j} \geq 1, \tag{4.27}$$

$$x_{i_1 i_2 j} = 1 \Rightarrow \sum_{i=1}^{m} x_{i_2 i j} \leq 1, \quad x_{i_1 i_2 j} = 1 \Rightarrow \sum_{i=1}^{m} x_{i_2 i j} \geq 1. \tag{4.28}$$

(4.27) and (4.28) each denote one equation by two inequalities. Moreover, it is necessary to ensure that no task $i_0 \in \{1, \ldots, m\}$ appears in more than one pair at the same (i.e. first or second) position:

$$\sum_{i=1}^{m} \sum_{j=1}^{n} x_{i_0 i j} = 1 \quad \text{and} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} x_{i i_0 j} = 1. \tag{4.29}$$

For simplicity, we have used the equation notation. Remember that, for concavity, we need to denote any equation by two inequalities. (4.29) also enforces that each task is indeed assigned to some machine. To be precise, if the chains are not to be regarded as cyclic, we need dummy tasks at the beginning and the end of each chain. Their definition and relation to the real tasks is obvious and thus left out here.

Additional constraints

Section 4.2.4 covers typical additional constraints. In fact, we have unary constraints whenever a task is not compatible with all machines. On the other hand, precedence constraints are natural, too, because one task may be based on the result of another task. In fact, suppose the machines are numbered according to their order in the assembly line. Then, for $j \in \{1, \ldots, n-1\}$, the tasks assigned to the j–th machine are processed before the tasks assigned to the $(j+1)$–st machine.

During the search

It is quite common to descend from one tree node to another one by putting restrictions on the distribution of individual items to individual bins. In the extreme case, an item i is fixed to some bin j. Then we can remove i and reduce c_j by a_i. On the other hand, if i is not fixed to a specific bin but merely restricted to a subset of bins, this is an example of unary constraints as discussed in Section 4.2.4.

Small number of bins

If n is too large for our purposes, we can unite several bins into one and obtain a less accurate, yet more efficiently evaluable model.

4.3.6 Delivery and Vehicle–Routing Problems

This class of optimization problems covers all cases in which service devices move items from service stations to delivery points. For example, in a delivery service, the service devices are vehicles, the delivery points are customers, the service stations are the company's storages, and the items are goods ordered by the customers. The vehicles may or may not be bound to specific service stations.

In the following, we will focus on the example of delivery services to make the explanations more illustrative. There are two different kinds of capacity constraints to consider. On one hand, each vehicle can only carry a certain volume and weight of goods. On the other hand, there may be temporal restrictions of all kinds: a maximum time for each tour, time windows for individual customers, temporal precedence constraints, etc. These two different kinds of constraints are very similar to the two different cases in load–balancing problems as discussed above: whether or not the reset times may be safely ignored.

In fact, the first kind of restriction can be simply handled by a Bin Packing Constraint in which the goods are the items, their weights or volume requirements are the item sizes, the vehicles are the bins, and the load capacities of the vehicles are the bin capacities. For the second kind of restriction, we have to apply the same trick as for reset times in load-balancing problems, namely we assign pairs of jobs to bins (=vehicles). Let $x_{i_1 i_2 j}$ again denote whether or not i_1 is processed immediately before i_2 by vehicle j. These variables allow the incorporation of conditions of the first kind: just introduce the weight or volume requirement of a good as a constant value depending on i_1, i_2, and j but only varying in i_1.

Sometimes the number of vehicles or tours of vehicles is not fixed but shall be kept to a minimum. This amounts to the first minimization problem discussed at the beginning of Section 4.3.

Additional constraints

Suppose that the vehicles are bound to fixed service stations. This simply means that a vehicle cannot process every task, which amounts to a set of unary constraints as before. Much like in the problem classes discussed before, the above–mentioned types of temporal constraints are easy to formulate as concave constraints.

During the search

It is common to descend from one tree node to another one by fixing the assignments of items to vehicles. One would also restrict items to time windows, which is again a unary constraint.

Small number of bins

If the number of vehicles is too large to have one bin for each vehicle, we are again able to form each bin from a set of vehicles.

4.3.7 Packaging and Partitioning

Packaging and partitioning problems often occur in a multi–dimensional setting. In the simplest case, each item $i \in \{1, \ldots, m\}$ has an extension a_{id} for each dimension d (that is, the items are brick–shaped, and the axis–parallel orientation of each brick is fixed). In general, the items may also be of arbitrary multi–dimensional shapes and may be put in the bin with a (more or less) arbitrary but axis–parallel stereometric orientation. For example, the problem of stapling furniture in a truck is an example of the three–dimensional case.

Clearly, with both, packaging and partitioning problems, nonlinear constraints appear inherently in form of geometric properties of objects or restrictions. For example in semiconductor manufacturing, there are usually circular wafers that have to be partitioned into rectangular dies.

One natural possibility is to apply several instances of the Bin Packing Constraint, one for each dimension. In *cutting problems*, where for example parts of clothes are to be cut out of a piece of textile, a natural choice of bins in one dimension would be to divide the area of the piece of textile into several two–dimensional stripes perpendicular to that dimension, each of which forms a bin. On the other hand, in the furniture problem, a natural choice would be to divide the available load volume into three–dimensional stripes. In the above case of bricks in fixed axis–parallel orientations, such an approach might yield quite a tight set of constraints in itself. In the more general case of arbitrary shapes and arbitrary two/three–dimensional orientations, we have to use the minimal diameter for each direction in order to ensure that the relaxation is correct.

However, in the furniture problem and many other three–dimensional problems, at least the alignment to the vertical direction is clear for many items. In such a case, the vertical extension of such an item may be taken as is, and the minimal horizontal extension is to be taken for the other two dimensions.

This model deals with *eccentric* items poorly. We distinguish two different types of eccentricity. The first type means that the ratio of the maximal and the minimal diameter of the item is particularly large. The second type means that the shape of the item deviates from a cuboid. Eccentricity of the second type is to be regarded as particularly high if the item is highly non–convex. Clearly, eccentricity of the first type yields poor bounds when the smallest diameter is taken in each dimension (if the two/three–dimensional orientation of the item is freely choosable). On the other hand, eccentricity

of the second type yields poor bounds because in our approach the item is replaced by its bounding box (the smallest diameter in all dimensions). Below, we will discuss a few techniques to overcome these problems to some extent during the search.

Additional constraints

In three–dimensional problems, we often have the restriction that certain items must not be put on certain other items, for example, heavy items should not be put on fragile items. For an item $i \in \{1, \ldots, m\}$, let a_{ix}, a_{iy}, and a_{iz} denote the *maximal* possible extensions in all three dimensions. Let the variables x_i, y_i, and z_i denote the reference point as to where the item is placed (e.g. the smallest values hit by the item in the respective dimension). Then the restriction that i_1 must not be put on top of i_2 can be formulated as a constraint on these variables, which is obviously concave:

$$\begin{aligned} x_{i_1} + a_{i_1 x} \geq x_{i_2} \wedge x_{i_1} \leq x_{i_2} + a_{i_2 x} \\ \wedge\, y_{i_1} + a_{i_1 y} \geq y_{i_2} \wedge y_{i_1} \leq y_{i_2} + a_{i_2 y} \\ \wedge\, z_{i_1} + a_{i_1 z} \geq z_{i_2} \wedge z_{i_1} \leq z_{i_2} + a_{i_2 z} \\ \Rightarrow z_{i_1} + a_{i_1 z} \leq z_{i_2} \end{aligned} \tag{4.30}$$

To combine this kind of restriction with a Bin Packing Constraint, we have to establish a relation between these variables and the variables that indicate the assignments of items to bins. For example, let the x–direction be divided into n stripes, where the i–th stripe ranges from $\bar{x}_{i-1}$ to $\bar{x}_i$, $i \in \{1, \ldots, n\}$. Then the following $2n$ additional concave constraints ensure the correct relation for i: $x_i \leq \bar{x}_j \Rightarrow \pi(i) \leq j$ and $x_i \geq \bar{x}_{j-1} \Rightarrow \pi(i) \geq j$ for all $j \in \{1, \ldots, n\}$.

During the search

In the multi–dimensional case, the reduction of the search space to a subset in a descending step in the search tree is typically realized by putting constraints on the positions and orientations of the items. A constraint on the position of an item is immediately translated into unary constraints on the possible assignments of this item to the bins. On the other hand, a constraint on the orientations of items means that we do not have to take the minimal diameter of this item for all dimensions, but we can use the real extensions of this item in all dimensions (or in a selection of dimensions if the constraints determine the orientation of the item only partially).

It is preferable to determine positions and orientations for the most eccentric items first. In fact, as we have seen, these are exactly the items for which taking the minimal diameter in all dimensions yields a particularly weak bound. As placing and orienting the most eccentric items first might be generally a good strategy in enumerative approaches, this does not collide with other algorithmic approaches.

In enumerative (and other) approaches to multi–dimensional packing problems, it is generally promising to combine items to *clusters*. This means that several items are regarded to be "glued" together in a fixed, rigid way to form a single, large item. Typically, eccentric items are combined such that the cluster is less eccentric than the individual items. For example, it is common to form clusters such that concave parts of items are filled in by convex parts of other clusters, or that several lengthy, thin "bars" are aligned to each other along the long side. Therefore, this strategy also helps weaken the problems with eccentric items.

Small number of bins

From the beginning of Section 4.3, recall that the number of bins should not be too large. The general approach to make bins from stripes of the available space gives full control over the balance of efficiency and accuracy, because, of course, a small number of stripes gives better efficiency, and a large number of stripes gives better accuracy.

4.3.8 Crew–Scheduling and Rolling–Stock Rostering

In the crew–scheduling problem, the crew staff of a public–transport company (e.g. an airline or a railroad company) are assigned to the individual connection services of the company. Analogously, in the rolling–stock rostering problem, some pieces of mobile equipment (carriages, planes, ...) are assigned to connection services.

For ease of exposition, we will focus on one concrete example, airline crew scheduling (the other examples of this problem type are analogous). Here, to each set of flight connection services staff is to be assigned to. More specifically, each flight connection has a certain number of vacant positions, and each position must be filled by a staff member. Each position requires specific skills, and the assigned crew member must have all of these skills. A crew member may be assigned to several positions if the flight connections do not overlap in time. However, a crew member may only be assigned a very limited number of positions between two rest periods.

There are at least two natural options for applying the Bin Packing Constraint to this kind of problem. For example, it might be natural to let the flight connections be the bins, and the crew members be the items. Likewise, it might be natural to let the crew members be the bins, and the positions be the items. In general, both options are certainly promising. However, neither option is appropriate for our algorithmic approach, because the number of bins is inherently large.

Therefore, our idea is a bit different: we do not consider crew members but working periods of crew members (the time between two rest periods). The first bin contains the position of an assigned crew member if this is the first position of the assigned crew member after a rest period. Analogously, the bin no. 2 contains a position if this is the second position after a rest period, and so on.

Additional constraints

There are a couple of side constraints such as time windows, minimum ground time, a maximum time crew are allowed to be away from their home base, etc. Moreover, there might be more complex rules, e.g. if a crew member is flying for more than 8 hours within a 24 hours period, the member is entitled to a longer rest period than usual. In order to allow more team oriented planning, there might be binding constraints that enforce that crew do stay together during all legs of a tour.

During the search:

We descend from one tree node to another one by fixing position assignments to the the working periods. We could reduce the search space dramatically by reducing domains taking into account the additional constraints.

Small number of bins

Due to the fact that the bins reflect working periods, and a whole tour does usually not take longer than 5 days, the number of bins is not too large. Usually, we would apply one Bin Packing Constraint per tour.

4.3.9 Resource and Storage Allocation Problems

Storage systems for example in computer networks may be composed of several storage devices having a wide variety of characteristics such as access time, bandwidth, volume, etc. The efficient management of a storage system is the central issue for a smooth operation of the network. We regard large networks consisting of many host computers and several application programs running concurrently on several computers. Each application requires a specific level of service from the storage system in order to allow flawless performance. The *Storage Allocation Problem* searches for an optimal distribution of service requests to storage devices.

In general, *Resource Allocation Problems* or *Storage Allocation Problems* can be seen as *vector packing problems*. This is actually a non–geometric variant of the multidimensional BIN PACKING problem: Each bin has several characteristics (dimensions) as volume, weight, bandwidth, etc. Each item requires a certain amount of a characteristic. This can be modeled easily by multiple instances of the Bin Packing Constraint, one for each characteristic.

The objectives arising from the problem are manifold. For example, one objective could be the maximization of service level or the performance of the system. Another one might be the minimization of acquisition or maintenance costs of the system. Objectives might be composed of convex functions, one for each characteristic. Nonlinear objectives might be replaced by a linear objective on an auxiliary variable, and an additional concave constraint incorporating the original objective. Likewise, the constraints may be convex or concave resource-usage functions, one for each activity.

Additional constraints

There might be resource constraints of any type. Clearly, the problem extends the well-studied case in which resource constraints are all linear. Nonlinear resource constraints are usually due to physical or financial circumstances. In the storage allocation problem in computer networks for example, nonlinearities are due to access speeds, cache sizes, the network topology, and the current utilization, only to mention a few. As all of the nonlinear functions arising in this context are usually either concave or convex, they can be modeled in terms of Concave Constraints.

During the search:

We descend from one tree node to another one by fixing resource assignments. The search space is shrunk dramatically by reducing the domains by propagating the resource constraints.

Small number of bins

Clearly, the number of bins might be quite large as the number of storage devices is. One idea is to merge multiple devices with the same characteristics to one bin. Thus, we can proceed from a comparatively small number of bins.

4.3.10 Financial Applications

Increased sophistication in financial models and the consideration of interacting phenomena lead to more and more detailed mathematical models that are often involving nonlinear constraints. For example, portfolio optimization has been quite an active research area applying linear and quadratic optimization techniques. The credit instruments and their risk management also involve nonlinear functions that are difficult or even impossible to formulate by means of linear or quadratic models. There might be various objective functions mainly such as the return maximization, the risk minimization, or arbitrary combinations from these.

In the *Portfolio Selection Problem* (MARKOWITZ, 1952), the total variance (risk) V of assets is to be minimized subject to a constraint that ensures a specified return R. The work in CHANG ET AL. (2000) extends this model by cardinality constraints that limit a portfolio to have a specified number K of asset types, and impose limits on the proportion of the portfolio held in a given asset. Clearly, this is done in order to obtain a diversification within the portfolio. While the original problem is easy to solve, the extended problem has been shown to be $\mathcal{NP}$–hard (BIENSTOCK, 1996). Clearly, the Portfolio Selection Problem can be seen as a HIGH MULTIPLICITY BIN PACKING PROBLEM (cf. Chapter 3) if we consider certain amounts of assets that might be held instead of proportions. This point of view is also more detailed as no fractional parts of assets may occur.

Our idea to impose the Bin Packing Constraint is the following: we partition the set of asset types into several categories, each of which is represented by a distinct bin. A natural categorization of assets might be given for example by the risk levels or their estimated return of investment. We impose restrictions on every single asset type which bin(s) it is allowed to be assigned to (namely the categories it fits in). In our model asset types might be restricted to more than one bin depending on the semantics of the chosen categorization.

We establish one dummy bin that absorbs all assets that are not chosen in the portfolio. The size of this bin is chosen such that it may absorb all item types in their multiplicities. Hence if $R = K = 0$, the trivial solution consists in the dummy bin holding all assets. Clearly, the risk of this solution is zero.

All other bins have a specific size that cannot be exceeded. This amounts to an upper bound on the number of assets that might be acquired in each category, and further on, to a bound on the number of assets that might be held of a specific type. Basically, this technique is related to the extension by the cardinality constraint, but much more general as the bounds are not imposed directly on the assets, but indirectly on categorized classes of assets. Furthermore, it is easily conceivable to apply more than one Bin Packing Constraint in order to categorize assets on the basis of multiple criteria.

In total, we have modeled the Portfolio Selection Problem by one or more Bin Packing Constraints, assignment restrictions which are clearly concave, and a quadratic objective function which can be modeled as a concave side constraint in support of a linear objective.

Additional constraints

There might be minimum transaction lots which can be modeled by linear inequalities which are clearly concave. Moreover, there are various nonlinear constraints such as convex transaction costs, nonlinear utilities and taxation costs. Furthermore, the estimation of volatility in terms of a nonlinear function plays a more and more important role in the detailed modeling of financial facts and circumstances.

During the search:

We descend from one tree node to another one by fixing certain types and multiplicities of assets to a portfolio or to the dummy bin.

Small number of bins

A small number of bins arises naturally from the categorization into specific classes.

4.4 Joint Evaluation of Bin Packing Constraints and Concave Constraints

4.4.1 Overview of the Preprocessing

We present a framework which is intended to be part of a search engine of a constraint programming package. Such a search engine first applies a preprocessing to prepare the data, then a search tree is traversed. Descending in the tree amounts to adding (usually linear) constraints to the constraint store. Thus, the approach is divided into a preprocessing stage and a search stage. During the preprocessing a certain data structure is constructed from the input values of the Bin Packing Constraint. This data structure will form an implicit representation of a superset of all feasible solutions to this instance of the Bin Packing Constraint. The original concave constraints might be used in the preprocessing for an initial feasibility check of the given instance. In the search stage, concave constraints that are held by the constraint store are evaluated during the search to compute certificates of infeasibility. Concave constraints may be added during the search to the constraint store without repeating the preprocessing stage.

Basically, the generation of the data structure during the preprocessing stage is again divided into two stages. The output of the first stage serves as an input to the second stage. These two stages are discussed in Sections 4.4.2 and 4.4.3, respectively. In Sections 4.4.4 and 4.4.5 additional concave constraints are generated from the properties of the original instance in order to relativize the effects of the relaxations to some extent.

Eventually, in Section 4.5, the joint evaluation with concave constraints is discussed and thus the 'big picture' of the algorithm is given.

4.4.2 Rounding of Input Values

First of all, we apply a rounding algorithm from Chapter 2 to the original instance data. Suitable for our purposes are the variants from Section 2.1.3 (Rounding to K rounding values) and Section 2.1.4 (Adaptive Rounding). The error measures should be chosen according to the specific application. In both cases, the parameters should be set as small as possible, but at the same time sufficiently large to keep the error at a reasonable scale. In practice, the parameter choice might be driven by the error bounding formulas from Sections 2.1.2 and 2.1.5.

Depending on the used technique, the output of the rounding procedure is the following:

- K rounding values $X_1, \ldots, X_K$, or
- K_1, and K_2, and K_1 pairs (X_ℓ, Δ_ℓ) of rounding values

Besides that, we have the information about which items of the original instance have been rounded to which rounding values.

4.4.3 Decoupling Relaxation

In the second preprocessing stage, the rounded instance from the above section is relaxed further if Adaptive Rounding was used. However, in Section 4.4.4 we will see that the effect of this relaxation can be undone to some extent.

We relax the exact correspondence between X– and the Δ–values. For each sequence $\ell \in \{1, \ldots, K\}$ of equidistant rounding values and for each bin $j \in \{1, \ldots, n\}$, we introduce two non-negative real–valued variables, $y_{\ell j}$ and $z_{\ell j}$. Thereby, $y_{\ell j}$ counts the number of items that were rounded down to the j–th rounding value in the ℓ–th equidistant sequence. In other words, they are now assigned to the j–th bucket according to Definition 2.11. Every item a_i that was rounded down to a rounding value $\tilde{a}_i = X_\ell + k \cdot \Delta_\ell$ with $k > 0$ contributes exactly k units to $z_{\ell j}$. More formally, we define

$$y_{\ell j} := \#\Big\{ i \;\Big|\; i \text{ is rounded down to } \tilde{a}_i = X_\ell + j \cdot \Delta_\ell \Big\}, \text{ and } z_{\ell j} := j \cdot y_{\ell j}, \tag{4.31}$$

considering all items $i \in \{1, \ldots, m\}$. For $\ell \in \{1, \ldots, K\}$, let d_ℓ denote the absolute contributions of X_ℓ–values:

$$d_\ell \;:=\; \sum_{j=1}^{n} X_\ell y_{\ell j}. \tag{4.32}$$

That is the number of items $i \in \{1, \ldots, m\}$ such that $\tilde{a}_i$ is in the ℓ–th sequence of equidistant rounding values, multiplied with the corresponding X_ℓ-value from the ℓ–th sequence. Analogously, let d'_ℓ denote the absolute contributions of Δ_ℓ–values:

$$d_{\ell'} \;:=\; \sum_{j=1}^{n} \Delta_{\ell'} z_{\ell' j}. \tag{4.33}$$

We are introducing this notation as we will use a non–integral relaxation in Section 4.4.6. Therefore, it is sufficient to keep the left hand sides from equations 4.32 and 4.33.

4.4.4 Reintroducing X–Δ–Correspondence

Recall that we have introduced the variables y_{ij} and z_{ij} to relax the correspondence between X– and Δ–values. The effect of this relaxation can be undone to some extent by means of introducing two kinds of additional concave constraints: For $i \in \{1, \ldots, m\}$ let $s_i \in \{1, \ldots, K\}$ and $t_i \in \{0, \ldots, R_k - 1\}$ such that $\tilde{a}_i = X_{s_i} + t_i \cdot \Delta_{s_i}$.

First, note that for $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$, a given value of y_{ij} imposes both, a lower as well as an upper bound on the possible values of z_{ij}. More specifically, the lower bound can be computed by adding up the values t_i from the following choice of items: Among all items $i \in \{1, \ldots, m\}$ such that $s_i = \ell$, first choose as many items $i \in \{1, \ldots, m\}$ as possible such that $t_i = 0$; next as many items as possible such that $t_i = 1$, next $t_i = 2$ and so on (the upper bound is computed analogously, starting from the items with t_i maximal; next t_i second maximal; and so on). Obviously, these two bounds are independent of j. So for $r := y_{ij}$, let L_{ir} and U_{ir} denote this lower and upper bound, respectively. Then the additional concave constraints of the first kind are these: For all $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$ and non-negative integral numbers r, we require

$$y_{ij} = r \implies L_{ir} \leq z_{ij} \leq U_{ir}\,. \tag{4.34}$$

A second kind of additional concave constraints addresses the obvious fact that not every value between L_{ir} and U_{ir} is possible for z_{ij} if $y_{ij} = r$. Suppose it is known that $h \in [L_{ir} \dots U_{ir}]$ is such a value. Then we can introduce the following disjunction as the second kind of additional concave function:

$$z_{ij} \leq h - 1 \ \vee \ z_{ij} \geq h + 1\,. \tag{4.35}$$

4.4.5 Reintroducing Integrality in Terms of Concave Constraints

As integrality of items in the original problem will be completely dropped by the relaxation in Section 4.4.6, we will introduce *relaxed integrality* by means of concave constraints in order to keep the problem as tight as possible. The integrality constraint $x_{ij} \in \{0, 1\}$ then translates to the constraint $x_{ij} \leq 0 \ \vee \ x_{ij} \geq 1$. As we now have to deal with multiplicities d_i of rounded items i, the original $x_{ij} \in \{0, 1\}$ constraint translates to a $x'_{ij} \in \{0, \dots, d_i\}$ constraint. Therefore, we add d_i constraints of the form

$$\begin{aligned} x_{ij} \leq 0 \quad &\vee \quad x_{ij} \geq 1 \\ x_{ij} \leq 1 \quad &\vee \quad x_{ij} \geq 2 \\ &\vdots \\ x_{ij} \leq d_i - 1 \quad &\vee \quad x_{ij} \geq d_i \end{aligned}$$

This has to be done for each bin $j \in \{1, \dots, n\}$. Note that there are now d_i constraints, but the size of the input, and therefore the number of x_{ij} variables shrinks by the same amount applying the rounding procedure from Section 4.4.2. Alternatively, integrality may be imposed by discrete variable domains.

4.4.6 Non-Integral Relaxation

In the following, we will drop the constraint that forces y_{ij} and z_{ij} to be integral. In other words, we allow y_{ij} and z_{ij} to assume arbitrary non-negative values. This is called the *non–integral relaxation*.

Now the relaxed instance from Section 4.4.3 is the new input for the non–integral relaxation. For notational convenience in the following, we will treat this relaxed instance as a new Bin Packing Constraint. Therefore, we introduce the following aliases: For $\ell \in \{1, \dots, K\}$ and $j \in \{1, \dots, n\}$, we set

$$a_\ell := X_\ell \ \text{ and } \ a_{\ell+K} := \Delta_\ell, \ \text{ and} \tag{4.36}$$

$$d_\ell \ := \ \sum_{j=1}^{n} X_\ell y_{\ell j} \ \text{ and } \ d_{\ell+K} \ := \ \sum_{j=1}^{n} \Delta_\ell z_{\ell j}. \tag{4.37}$$

That is, a_ℓ is the X_ℓ–value in the ℓ–th interval, and $a_{\ell+K}$ is the Δ_ℓ–value, d_ℓ is the sum of X_ℓ–contributions multiplied by X_ℓ, and $d_{\ell+K}$ is the sum of Δ_ℓ–contributions multiplied by Δ_ℓ. If another rounding procedure than Adaptive Rounding is used, the new instance has only size K, and the Δ_ℓ–values are obsolete. When the Adaptive Rounding procedure from Section 4.4.2 and the decoupling

relaxation from Section 4.4.3 have been used, the size of the instance has decreased by orders of magnitude. Therefore, we have to set

$$M := 2 \cdot K \text{ and } N := n, \tag{4.38}$$

and M corresponds to the number of item sizes and N to the number of bins in the reduced instance.

Now, we use this instance to compute the polytope of the non-integral solution space, i.e. we allow non-integral distributions of items to bins.

4.4.7 Greedy enumeration of the non-integral solution space

For the following lemma, recall the assignments from (4.37). Also recall that d_{i_0} is the number of items of type i_0 multiplied by their size a_{i_0}, and c_{j_0} is the capacity of bin j_0. The lemma states that – at every extremal point of the non–integral solution space – either a bin reaches its capacity limit, or an item was completely distributed, or both.

Lemma 4.8. *Consider an instance of the non–integral relaxation, and let $x \in \mathbb{R}^{M \times N}$ be a vertex of the non–integral solution space. Then there are $i_0 \in \{1, \ldots, M\}$ and $j_0 \in \{1, \ldots, N\}$ such that at least one of the following two conditions is fulfilled:,*

$$a_{i_0} \cdot x_{i_0 j_0} = d_{i_0}, \tag{4.39}$$

$$a_{i_0} \cdot x_{i_0 j_0} = c_{j_0}. \tag{4.40}$$

Proof. For notational convenience in this proof, we will write $x(i,j)$ instead of x_{ij} in the following. Let x_0 be a vertex of the non–integral solution space. In this proof, we will consider a certain undirected auxiliary graph $G = (V, E)$, which is related to x_0. In that, V is the set of all pairs (i,j) such that $x(i,j) > 0$, where $i \in \{1, \ldots, M\}$ and $j \in \{1, \ldots, N\}$. There is an edge in E connecting $(i_1, j_1) \neq (i_2, j_2)$ if, and only if, $i_1 = i_2$ or $j_1 = j_2$. If $i_1 = i_2$, this edge is called a *type–1 edge*, otherwise (that is, if $j_1 = j_2$), it is called a *type–2 edge*. We will call a path p in G *special* if the type–1 and type–2 edges appear on this path alternatingly (in other words, no two edges of the same type appear on p immediately after each other). A *special cycle* is a cycle such that there is at most one pair of successive edges of the same type (type 1 or type 2). The intuition behind this definition of special cycles is a special path closed to a cycle at some node, and if the path has odd length, two successive edges of the same type arise. As usual, a path or cycle is called *elementary* if no node is hit more than once.

In the following, we will apply a major case distinction. First, we consider elementary special cycles and elementary inclusion–maximal special paths with an *even* number of nodes; second, elementary special cycles and inclusion–maximal elementary special paths with an *odd* number of nodes. Clearly, at least one of these cases is met.

So, for the first major case, consider an elementary special cycle or inclusion–maximal elementary special path with an *even* number of nodes. Let $(i_1, j_1), \ldots, (i_{2k}, j_{2k})$ denote the nodes of this path or cycle in the order of occurrence. For $\varepsilon > 0$, let x'_ε and x''_ε be defined by:

$$x'_\varepsilon(i_{2\ell-1}, j_{2\ell-1}) := x(i_{2\ell-1}, j_{2\ell-1}) + \varepsilon, \tag{4.41}$$

$$x'_\varepsilon(i_{2\ell}, j_{2\ell}) := x(i_{2\ell}, j_{2\ell}) - \varepsilon, \tag{4.42}$$

$$x''_\varepsilon(i_{2\ell-1}, j_{2\ell-1}) := x(i_{2\ell-1}, j_{2\ell-1}) - \varepsilon, \tag{4.43}$$

$$x''_\varepsilon(i_{2\ell}, j_{2\ell}) := x(i_{2\ell}, j_{2\ell}) + \varepsilon, \text{ for all } \ell \in \{1, \ldots, k\}; \tag{4.44}$$

$$x'_\varepsilon(i,j) := x(i,j), \text{ and} \tag{4.45}$$

$$x''_\varepsilon(i,j) := x(i,j), \text{ for all other } (i,j) \in V. \tag{4.46}$$

Since x_0 is a convex combination of x'_ε and x''_ε, x_0 cannot be a vertex of the non–integral solution space if both x'_ε and x''_ε are feasible. Therefore, it suffices to show that both x'_ε and x''_ε are feasible for $\varepsilon > 0$ small enough, unless either option of Lemma 4.8 is fulfilled by some node of G. Obviously, for a special cycle with an even number of nodes, this is true for every $\varepsilon > 0$ that is small enough such that x'_ε, $x''_\varepsilon \geq 0$. In case of a special path with an even number of nodes, the same argument applies except that there may be a problem at the end-nodes. Thus, we will focus on the end-nodes of such an inclusion–maximal special path, (i_1, j_1) and (i_{2k}, j_{2k}). It is either $i_1 = i_2$ and $i_{2k-1} = i_{2k}$, or it is $j_1 = j_2$ and $j_{2k-1} = j_{2k}$. If $j_1 = j_2$ and $j_{2k-1} = j_{2k}$, the inclusion–maximality of the path enforces $x(i_1, j) = 0$ for all $j \in \{1, \ldots, M\} \setminus \{j_1\}$. However, then the first condition of the formal problem description in the introduction implies that the first option of Lemma 4.8 is fulfilled by (i_1, j_1) (and also by (i_{2k}, j_{2k})). On the other hand, if $i_1 = i_2$ and $i_{2k-1} = i_{2k}$, the inclusion–maximality of the path enforces $x(i, j_1) = 0$ for all $i \in \{1, \ldots, M\} \setminus \{i_1\}$. Now, if $a_{i_1} \cdot x(i_1, j_1) = c_{j_1}$ or $a_{i_{2k}} \cdot x(i_{2k}, j_{2k}) = c_{j_{2k}}$, the first option of Lemma 4.8 is again fulfilled by (i_1, j_1) or (i_{2k}, j_{2k}). Otherwise (that is, $a_{i_1} \cdot x(i_1, j_1) < c_{j_1}$ and $a_{i_{2k}} \cdot x(i_{2k}, j_{2k}) < c_{j_{2k}}$), ε can obviously be chosen such that x'_ε and x''_ε do not violate the capacity constraints and are non-negative (for the latter point, recall that $x(i_1, j_1)$, $x(i_{2k}, j_{2k}) > 0$ due to the definition of G). This concludes the first major case.

For the second major case, consider an elementary special cycle or inclusion–maximal elementary special path with an *odd* number of nodes. Let $(i_1, j_1), \ldots, (i_{2k+1}, j_{2k+1})$ denote the nodes of this path or cycle in the order of occurrence.

First we will consider cycles, and paths only later on. In case of a cycle, there are two successive edges of the same type. Without loss of generality, (i_{2k+1}, j_{2k+1}) is the node in between these two edges. Then it is $i_{2k} = i_{2k+1} = i_1$ or $j_{2k} = j_{2k+1} = j_1$. In either case, the nodes $(i_1, i_2), \ldots, (i_{2k}, j_{2k})$ form an elementary special cycle on an even number of nodes. This case has already been considered in the first major case.

So, for the rest of the proof, we will focus on an inclusion–maximal elementary special path with an odd number of nodes, $(i_1, j_1), \ldots, (i_{2k+1}, j_{2k+1})$. Then it is $j_1 = j_2$ or $j_{2k} = j_{2k+1}$. Without loss of generality, we assume $j_1 = j_2$. Since the path is inclusion–maximal, it is $x(i_1, j) = 0$ for all $j \in \{1, \ldots, M\} \setminus \{i_1\}$. Therefore (analogously to some other subcase above), the first condition of the formal problem description in the introduction implies that the first option of Lemma 4.8 is fulfilled by (i_1, j_1). This completes the second major case and thus the proof of Lemma 4.8. □

Algorithm 5 computes all vertices of the non–integral solution space polytope using a greedy strategy. The following theorem generalizes a folklore result for the KNAPSACK PROBLEM using a greedy algorithm and dropping integrality of items.

Theorem 4.9. *Consider an instance of the non–integral relaxation and let x be a vertex of its solution space. Then there is a total ordering "$\prec$" on $\{1, \ldots, M\} \times \{1, \ldots, N\}$ such that x is the result of the above greedy–like strategy.*

Proof. Formally, we will prove Theorem 4.9 for a slightly more general problem, namely, there may also be conditions of the form "$x_{ij} = 0$" for pairs $i \in \{1, \ldots, M\}$ and $j \in \{1, \ldots, N\}$. Clearly, the original problem is the special case in which the number of these additional conditions is zero.

We will prove Theorem 4.9 by induction on the number of pairs $i \in \{1, \ldots, M\}$ and $j \in \{1, \ldots, N\}$ such that there is *no* condition "$x_{ij} = 0$". Clearly, if this number is zero, the claim is trivial. So suppose there is at least one pair (i, j) such that "$x_{ij} = 0$" is *not* required.

For the induction step, we will utilize Lemma 4.8, that is, we will utilize the proven existence of a value $x_{i_0 j_0}$ as guaranteed by Lemma 4.8. More specifically, we will do that as follows: we let (i_0, j_0) be the first pair to be considered by the greedy algorithm, that is, $(i_1, j_1) := (i_0, j_0)$. To apply the induction hypothesis, we modify the instance slightly. Let d'_i and c'_i be defined by $d'_{i_0} := d_{i_0} - a_{i_0} x_{i_0 j_0}$, $c'_{j_0} := c_{j_0} - a_{i_0} x_{i_0 j_0}$, $d'_i := d_i$ for all $i \in \{1, \ldots, M\} \setminus \{i_0\}$, and $c'_j := c_j$ for all $j \in \{1, \ldots, N\} \setminus \{j_0\}$.

Algorithm 5 `GreedyComputeVertices`

check initial feasibility:
if $\sum_{i=1}^{M} d_i > \sum_{j=1}^{N} c_j$ **return** "infeasible"
if there is one item that does not fit any bin **return** "infeasible"

initialize:
introduce x and $x_k \in \mathbb{R}^{M \times N}$ for $k \in \{1, \ldots, (MN)!\}$.
calculate all orderings $\prec_k$ on $\{1, \ldots, M\} \times \{1, \ldots, N\}$.

for $k := 1, \ldots, (MN)!$ **do**
$x \leftarrow 0$
forall (i, j) in $\prec_k$ **do** (in this order):
repeat increase x_{ij} **until**

$$\sum_{j=1}^{N} a_i \cdot x_{ij} = d_i \quad || \quad \sum_{i=1}^{M} a_i \cdot x_{ij} = c_j. \tag{4.47}$$

$x_k \leftarrow x$
return x_k **forall** $k \in \{1, \ldots, (MN)!\}$

Moreover, let x' be defined by $x'_{i_0 j_0} := 0$ and $x'_{ij} := x_{ij}$ for all $(i, j) \neq (i', j')$. To prove the induction step, it now suffices to show that x' forms a vertex of the modified non–integral solution space with respect to d' and c'. Suppose for a contradiction that x' is not a vertex. Then there is a value $x''_{i,j}$ for each $i \in \{1, \ldots, M\}$ and $j \in \{1, \ldots, N\}$ such that $x' + x''$ and $x' - x''$ are feasible with respect to d' and c' and at least one of the components of x'' is non–zero. Obviously, $x - x''$ and $x + x''$ are then feasible with respect to d and c. In summary, x would not be a vertex either. This contradiction proves Theorem 4.9. □

Corollary 4.10. *For any vertex of the non–integral solution space, at most $M + N - 1$ variables are non–zero, and at most $2N - 2$ variables are fractional.*

Proof. Algorithm 5 distributes items $\{1, \ldots, M\}$ to bins $\{1, \ldots, N\}$ for all orderings $\{1, \ldots, M\} \times \{1, \ldots, N\}$ in such way that either an item gets completely distributed, or a bin meets the capacity constraint. This might occur at most $M + N - 1$ times. When using the above greedy approach, it might happen that at most 2 items per bin are splitted. For the first and the last bin, there is only one fractional item. □

4.4.8 Efficient enumeration of the non-integral solution space

Clearly, this naive enumeration approach has quite a bad performance. Simply applying Algorithm 5 to all possible orderings, would yield to $(MN)!$ orderings! However, that number can be decreased dramatically in practice, as Algorithm 5 calculates one and the same vertex many times. We will develop a strategy that avoids computing duplicates and generates every vertex of the solution space exactly once.

In the following, we will denote by a *partial solution* a matrix $x_k \in M \times N$ that has not completed the run through one ordering $\prec_k$ in the above algorithm. In other words, this refers to a set of vertices, of which all have the same non-zero entries in common. In the following, we will see, that identical

partial solutions always cause identical sets of vertices, so we need an efficient strategy to eliminate all but one. We will use a tree T to describe all orderings leading to pairwise different (partial) solutions x. Every node of T represents a pair $(i,j) \in \{1,\dots,M\} \times \{1,\dots,N\}$. The root node has $M \cdot N$ direct successors, namely all pairs $\{1,\dots,M\} \times \{1,\dots,N\}$. For notational convenience, we order these nodes lexicographically from left to right. Every succeeding node v of those has $M \cdot N - 1$ direct successors, namely all pairs $\{1,\dots,M\} \times \{1,\dots,N\}$ except itself. All of these have again $M \cdot N - 2$ direct successors, and so on. Again, all direct successors of a node v within the tree are ordered lexicographically. Then, a path of length h from the root node ending at a node v with height h implies a sequence of h pairwise-different pairs from $\{1,\dots,M\} \times \{1,\dots,N\}$. This path implicitly describes a partial solution that we will denote by x_v. The path P_v unto a leaf v of T then describes an ordering on $\{1,\dots,M\} \times \{1,\dots,N\}$. Therefore, the polytope P of the non-integral solution space is completely defined by the leaves of T. The resulting maximal tree height is then $M \cdot N$. Anyway, we can decrease this height to $M + N - 1$ using the following deliberation.

Consider a tree node v and the corresponding (partial) solution x_v. The greedy algorithm increases an entry at position (i,j) of x_v such that either item i fits completely into bin j, or bin j is completely filled. In the first case, all zero entries in the i-th row can be *fixed* to zero as the item i is completely distributed. In the latter case, the zero entries in the j-th column are fixed to zero as there is no space in the j-th bin for any items of another type. Therefore, all successor nodes labeled $(i,\star)$ or $(\star,j)$, respectively, can be ignored as those rows or columns, respectively, are fixed to zero and the next node labeled (k,l) in the corresponding subtree that delivers $x_{kl} > 0$ is already a direct successor of v, as (k,l) cannot be part of the path to P_v. Using this strategy, the tree has at most height $M+N-1$. Therefore, in the following we will merely consider this compressed tree.

Observation 4.11. *Any path of length h in T from the root node to a node v corresponds to a partial solution $x_v \in \mathbb{R}_+^{M \times N}$ where exactly h entries are non-zero in a one-to-one fashion. In particular, T has at most height $M + N - 1$.*

Theorem 4.12. *Let v and w be two nodes within T such that $x_v \equiv x_w$. Then the subtrees T_v rooted at v and T_w rooted at w contain the same vertices of the non–integral solution space.*

Proof. As $x_v \equiv x_w$, v and w have the same height h within T, which reflects exactly the number of non-zero entries in x_v or x_w, respectively. The non-zero entries must have been generated by the same tuples, but in a different order. Therefore, exactly the same columns and rows have been fixed to zero. Finally, the set of tuples to be still processed in the subtrees rooted at v or w, respectively, is identical in both nodes. As the nodes are ordered lexicographically, the subtrees T_v and T_w are exactly identical. □

That gives rise to a strategy in order to eliminate nodes leading to duplicate solutions. As described above, we maintain a tree which is lexicographically ordered in every node. We will construct the tree from the root via depth-first search (DFS). Whenever we encounter a node while descending in a subtree that leads to a solution that has been already computed by a lexicographically smaller node (graphically speaking a node or path, respectively, further left within the tree) the node is cut off. For that reason, we have to find a strategy how to efficiently check for a given tree node v whether $x_v \equiv x_w$ for a lexicographically smaller w. Let P_v be the path from the root node to a node v. We say P_v is constructed from a path P_w of the same length by a (k,ℓ)*–move*, if the pair at position k in P_v is moved to the position ℓ, and the relative order of all other pairs is kept.

Lemma 4.13. *Let v and w be two nodes in the same layer h of T. Let P_w arise from P_v by a (k,ℓ)–move, and let (i_0,j_0) denote the moved pair (that is, (i_0,j_0) is at position k in P_v and at position ℓ in P_w). If $x_{i_0 j_0}$ in the partial solution x_v corresponding to v equals $x_{i_0 j_0}$ in the partial solution x_w corresponding to w it is $x_v \equiv x_w$.*

Proof. Without loss of generality, suppose $k > \ell$ (due to the perfect symmetry of the moving operation and the claim of Lemma 4.13, the claim follows immediately for $k < \ell$). Clearly, x_{ij} in x_v equals x_{ij} in x_w for all pairs at positions $1, \ldots, \ell - 1$.

Next consider the pairs (i, j) that are at positions $\ell, \ldots, k - 1$ in P_v. We may assume $x_{ij} > 0$ in x_v because otherwise the claim is evident. The assumption $x_{i_0 j_0}$ in x_v is equal $x_{i_0 j_0}$ in x_w means that the values x_{ij} in x_w of all of these pairs do not have any effect on the value $x_{i_0 j_0}$ in x_w. Since $x_{i_0 j_0} > 0$ in x_w, this is only possible if it is $i \neq i_0$ and $j \neq j_0$ for any of these pairs (i, j) that fulfills $x_{ij} > 0$ in x_w. For the pairs (i, j) such that $x_{ij} = 0$ in x_w, $x_{ij} = 0$ in x_v follows immediately from the fact that the x–value of a pair (i, j) cannot increase through moving it one step forward. On the other hand, a set of pairs (i, j) such that $i \neq i_0$ and $j \neq j_0$ is obviously not affected, either, by the value of $x_{i_0 j_0}$ in x_w if all pairs (i, j) except for (i_0, j_0) that participate in the permutation either fulfill $i \neq i_0$ and $j \neq j_0$ or $x_{ij} = 0$ in x_v and $x_{ij} = 0$ in x_w. Therefore, it x_{ij} in x_v equals x_{ij} in x_w for all of these pairs (i, j) as well.

Finally, consider the pairs (i, j) at positions $k + 1$, $k + 2, \ldots$ Since x_{ij} in x_v equals x_{ij} in x_w for all pairs (i, j) at positions $1, \ldots, k$, and since all permutations took place within $(1, \ldots, k)$, the greedy algorithm performs exactly the same steps $k + 1$, $k + 2, \ldots$ for P_v and P_w. □

Theorem 4.14. *While descending in the search tree T from a node v at height h to a direct successor w at height $h + 1$, a pair (i, j) is added to the sequence P_v. Then one of the following cases occurs:*

(1) the pair (i, j) is lexicographically greater than any other pair in P_v: node w is saved and x_w calculated

(2) the pair (i, j) is lexicographically smaller than the lexicographically greatest pair in P_v and

- *(a) there is no pair $(i, \star)$ or $(\star, j)$ in P_v: this partial solution has already been computed by a lexicographically smaller ordering, thus node w can be dropped.*
- *(b) there is one or more pair $(i, \star)$ or $(\star, j)$ in P_v: if (i, j) can be inserted at any position $1, \ldots, h$ within P_v always resulting in the same solution x_w as if (i, j) was placed at position $h + 1$, the node w can be dropped. Otherwise, it must be saved.*

Applying this procedure does not miss any vertex of the polytope P of the non-integral solution space.

Proof. As we always save the lexicographically smallest node w leading to a (partial) solution x_w, case (1) exactly meets this claim. If the other case is encountered, we have to determine, if an insertion of (i, j) at positions $1, \ldots, h$ within P_v is able to generate the same (partial) solution x_w as the ordering in P_w. This is easily affirmed, if there are no pairs $(i, \star)$ or $(\star, j)$ in P_v that may result in different entries in row i or column j of x_w, whenever (i, j) is inserted at a lower position than $h+1$. Therefore, the solution x_w has been calculated by a lexicographically smaller ordering before, namely that one, where (i, j) is inserted in P_v before the next lexicographically greater item than (i, j). Thus, node w can be dropped and the corresponding subtree does not need to be further explored. If there are pairs $(i, \star)$ or $(\star, j)$ in P_v, that may "interfere" with an insertion of (i, j) in P_v at a position lower than $h+1$, we have to check whether there is a lexicographically smaller ordering producing exactly the same solution x_w. As Lemma 4.13 states, we have to diagnose only permutations coming about by insertion of (i, j) into P_v as other permutations would have been already checked further up in the tree. For the same reason as above, it even suffices to check only permutations, where (i, j) is inserted before any pair $(i, \star)$ or $(\star, j)$. This can be easily done by applying the greedy strategy from Algorithm 5 to P_v starting at the root node: whenever a pair $(i, \star)$ or $(\star, j)$ occurs in the path, it is checked, whether the insertion of (i, j) at the position immediately before a considered pair $(i, \star)$ or $(\star, j)$, respectively, results in the same entries $x_{i\star}$ or $x_{\star j}$, respectively, as in x_w for all $\star$. If we can find a pair $(i, \star)$ or $(\star, j)$, respectively, for which this is true, we can immediately drop the node as Theorem 4.12 states that all

solutions contained in the subtree rooted at that node have been explored before. Thus, by applying this procedure, all pairwise-different vertices of the non-integral solution space are computed exactly once, in particular, no vertex is missed. □

Algorithm 6 `lexiSmaller(Queue` P`, bool` $item$`, bool` bin`)`

```
if P.size()=1 then return false
t ← P.lastElement()
greater ← correspond ← iteminsert ← bininsert ← false
forall (i, j) in P\{t} do
    if (t.i = i) then
        correspond ← true
        if (t < (i, j)) then
            greater ← true
            if (bin && !iteminsert) then
                return true
        iteminsert ← true
    else if (t.j = j) then
        correspond ← true
        if (t < (i, j)) then
            greater ← true
            if (item && !bininsert) then
                return true
        bininsert ← true
    else if (t < (i, j)) then
        greater ← true
    if (!correspond && t < (i, j)) then
        return true
if (!greater) then
    return false
if (!correspond) then
    return true
return false
```

We summarize the above results in an algorithm for traversing the search tree. In this algorithm, we use a function `lexiSmaller()` to check, whether a solution x, that arises from a path P within the tree has already been computed earlier by a lexicographically smaller ordering. The procedure returns false, if no such ordering exists, true otherwise. (1) and $(2a)$ from Theorem 4.14 can be tested straightforwardly for each path P. $(2b)$ can be checked by the following test: if there are any corresponding $(i, \star)$ or $(\star, j) \in P$, it must be ensured, that (i, j) can be inserted into P before one of these tuples without changing x_{ij} nor $x_{i\star}$ or $x_{\star j}$, respectively. This is exactly the case if either item i is getting empty or bin j is getting full. If an item i gets empty adding (i, j) to P, and there is a lexicographically greater $(\star, j) \in P$, (i, j) can be inserted before $(\star, j)$ without changing x_{ij} nor $x_{\star j}$, because the item gets empty in the corresponding bin. Analogously, if a bin j gets full adding (i, j) to P, and there is a lexicographically greater $(i, \star) \in P$, (i, j) can be inserted before $(i, \star)$ into P without changing x_{ij} nor $x_{i\star}$, because the bin got full. The complete procedure can be done in $\mathcal{O}(M + N - 1)$ and is described in Algorithm 6.

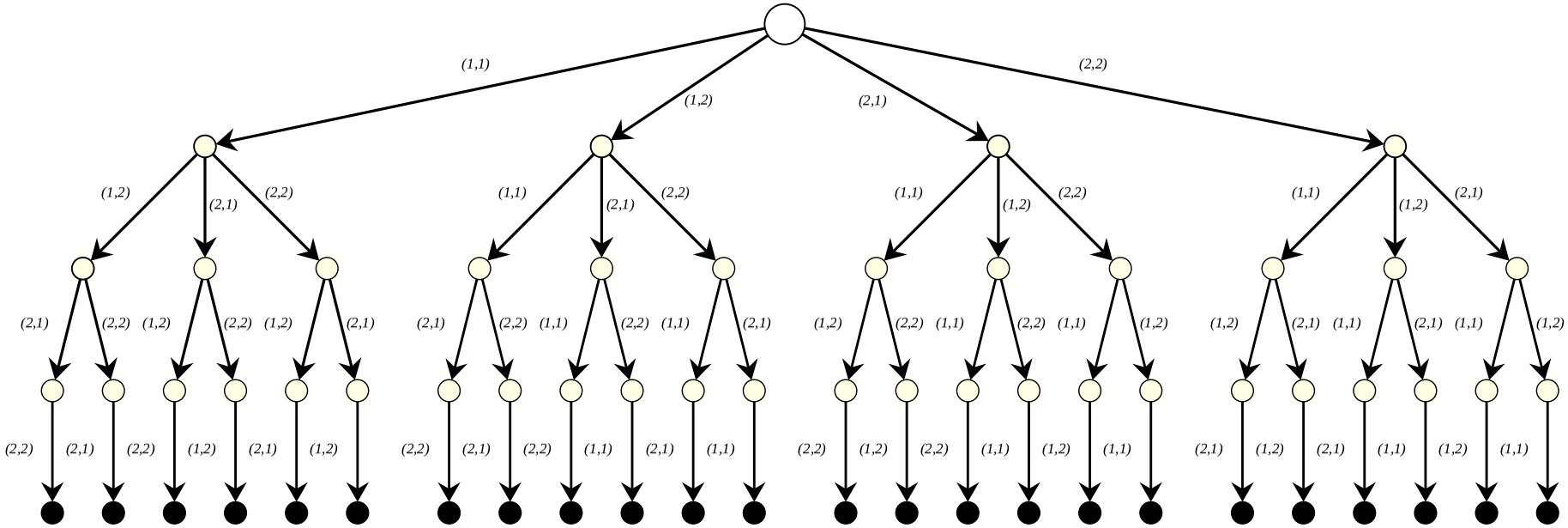


Figure 4.1: Full search tree generated by all orders of tuples $(i, j) \in \{1, 2\} \times \{1, 2\}$. This is the output of the greedy strategy from Algorithm 5. Each edge (i, j) carries the weight of the (fractional) amount of item i put into bin j in this special distribution.

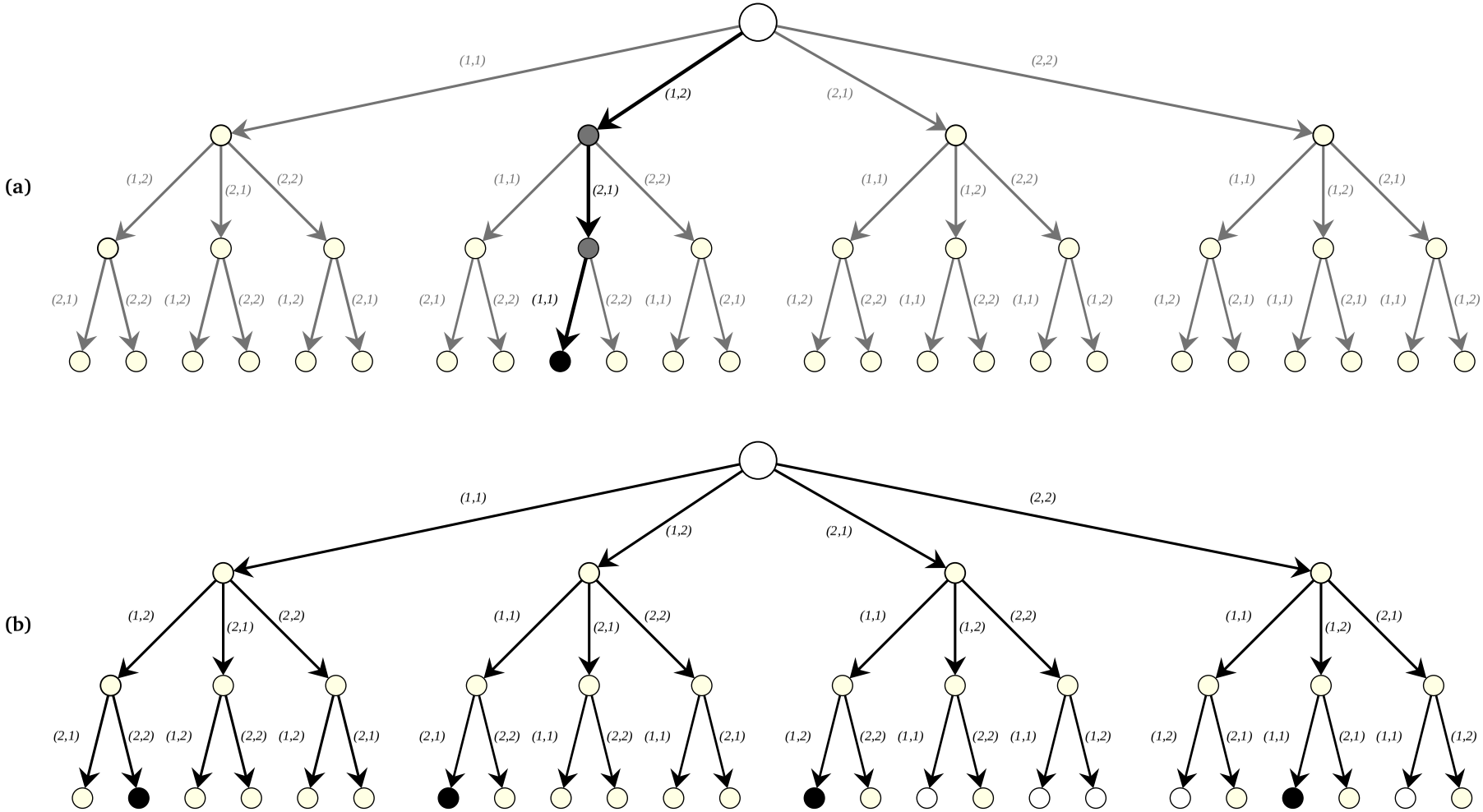


Figure 4.2: (a) In each layer either a bin gets full or an item is completely distributed to bins. So the search tree can be compressed to have at most height $M + N - 1$. Other edges have a zero value and thus can be dropped. A path to a vertex of the polytope P of the non-integral solution space consists of at most $M + N - 1$ edges, e.g. the path $(1, 2) \to (2, 1) \to (1, 1)$ is highlighted. **(b)** Many vertices are computed more than once using Algorithm 5. In our example, only the black vertices are relevant. From each relevant vertex, only the lexicographically smallest, i.e. graphically speaking the leftmost vertex is saved. All other vertices may be dropped. For example, the marked path from Figure (a) has been already computed by the path $(1, 2) \to (1, 1) \to (2, 1)$ which has a lexicographically smaller ordering. So it is dropped.

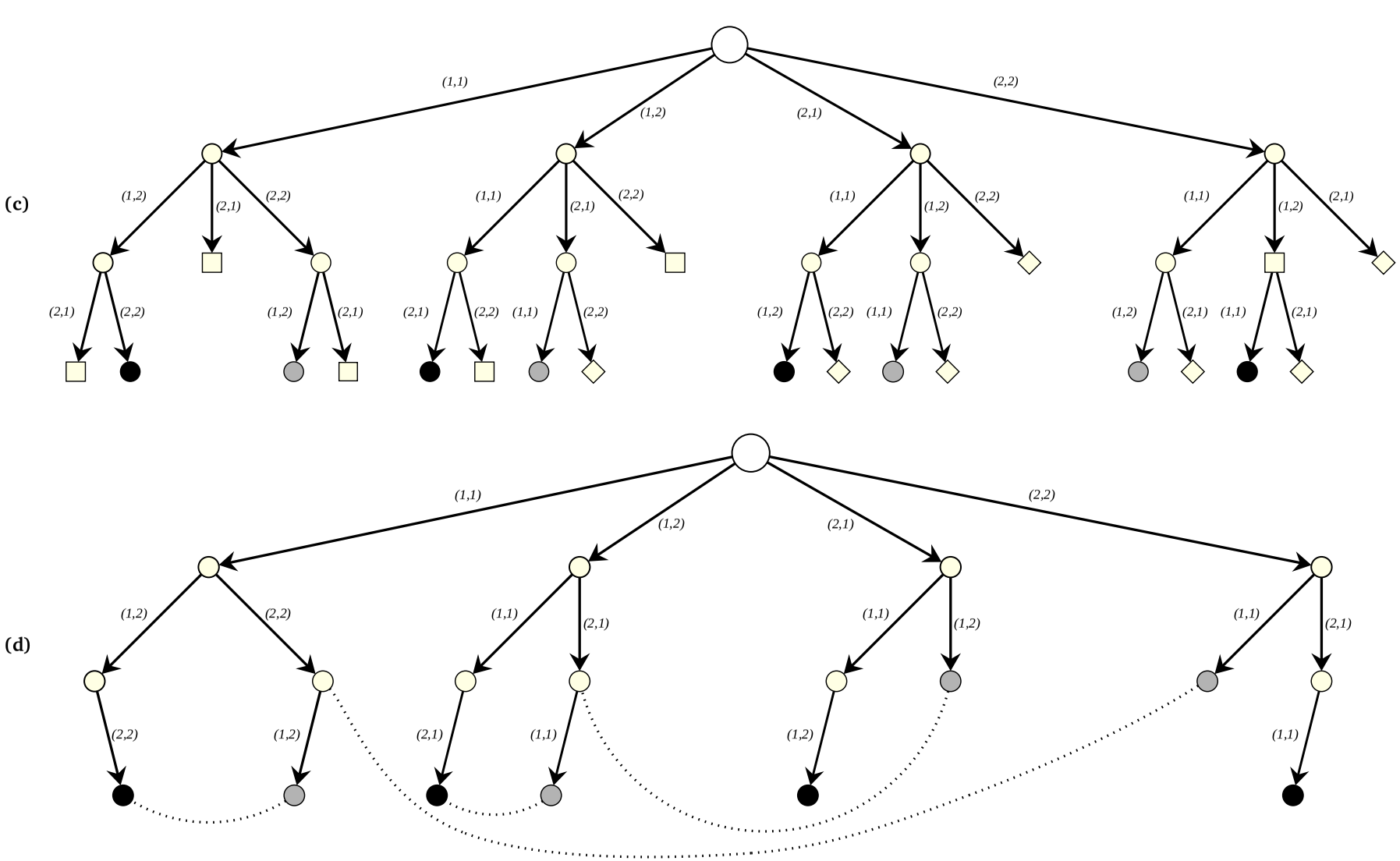


Figure 4.2: (c) Starting from (b), we assume that item 1 does not completely fit into bins 1 or 2, whilst item 2 does. So, item 1 is always distributed over 2 bins. In every layer of the tree, either a bin is already full (□) or an item is completely distributed (♢). These subtrees are cut off in **(d)**. Grey nodes have been calculated by a lexicographically smaller ordering (a node further left in the tree).These nodes are connected to their lexicographically smaller equivalents by the dotted curves. This might occur in any layer of the tree.

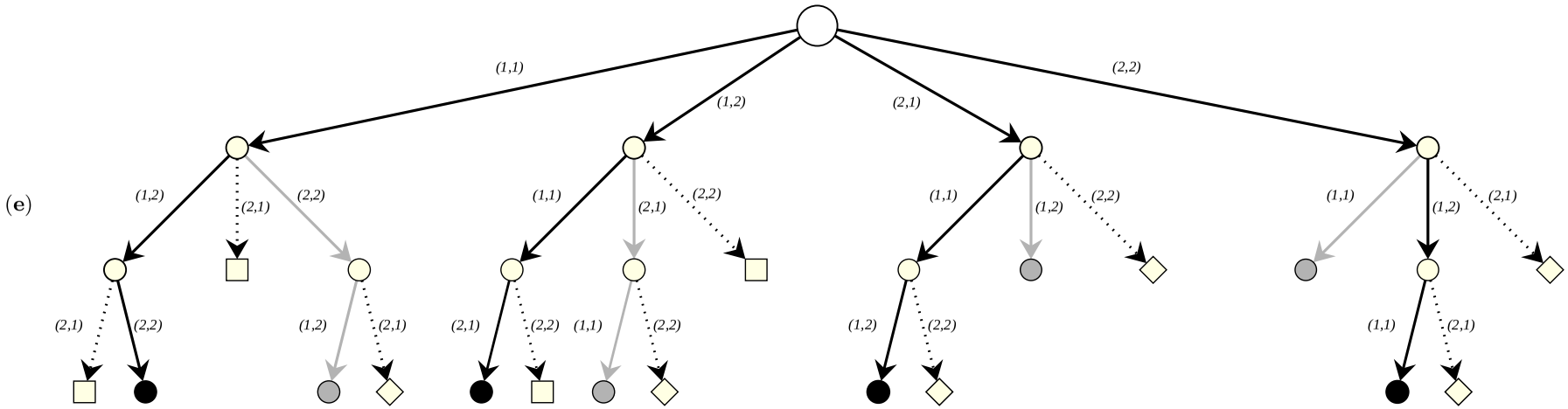


Figure 4.2: (e) The tree traversal algorithm (cf. Algorithm 7) takes (c) and (d) into account. The tree is traversed by depth fist search (DFS) from the left to the right. Lexicographically smaller nodes are automatically cut off during the search. The dotted branches are cut off due to the fact that either there is no capacity in the respective bin left (illustrated by a □), or an item is completely distributed yet (illustrated by a ♢). The lexicographically smallest relevant solutions are depicted by the black nodes. They are saved in a solution set. The grey branches symbolize (partial) solutions that have been already generated by a branch further left in the tree, so they can be cut off safely. For example, the leftmost grey branch is the path $(2,2),(1,2)$. Clearly, due to the fact that in the father node of that path, bin 1 is completely filled yet, is does not matter, if bin 2 is filled with item 1 first, and then item 2, or vice versa. So the solutions are equivalent, and it is sufficient to save the leftmost of them.

Algorithm 7 `TreeTraversal(Array` c`, Array` d`, Matrix` x`, Queue` P`, Queue` Q`)`

Given: $c \in \mathbb{R}_+^M, d \in \mathbb{R}_+^N$
Start: $x \in \mathbb{R}_+^{M \times N} \equiv 0$, Queues P empty, $Q \leftarrow \{1, \ldots, M\} \times \{1, \ldots, N\}$

initialize:
if $P.isEmpty()$ **then forall** $(i, j) \in Q$ **do**
    `TreeTraversal`$(c, d, x, \{(i, j)\}, Q)$

item $\leftarrow$ bin $\leftarrow$ false
$(i, j) \leftarrow$ P.lastElement()
if $(c_j \leq d_i)$ **then**
    $x_{ij} \leftarrow c_j$
    bin $\leftarrow$ true
if $(c_j \geq d_i)$ **then**
    $x_{ij} \leftarrow d_i$
    item $\leftarrow$ true
if `lexiSmaller`$(P, item, bin)$ **then**
    $x_{ij} \leftarrow 0$
    return
$d_i \leftarrow d_i - x_{ij}$
$c_j \leftarrow c_j - x_{ij}$
if $(d_i \leq c_j)$ **then**
    delete$(Q, (i, \star))$
if $(d_i \geq c_j)$ **then**
    delete$(Q, (\star, j))$
if $Q.isEmpty()$ **then**
    save(x)
else forall $(i, j) \in Q$ **do**
    `TreeTraversal`$(c, d, x, P \cup \{(i, j)\}, Q)$
$c_j \leftarrow c_j + x_{ij}$
$d_i \leftarrow d_i + x_{ij}$
$x_{ij} \leftarrow 0$
return

Figure 4.1 depicts a full search tree for the case $(i, j) \in \{1, 2\} \times \{1, 2\}$ as it is delivered by the simple greedy strategy of Algorithm 5. Each edge (i, j) carries the weight of the (fractional) amount of item i put into bin j. Figure 4.2 illustrates Algorithm 7: in (a) it is shown that in each iteration of the algorithm either a bin gets completely filled, or an item is completely distributed. Therefore, the maximum height of the search tree is at most $M + N - 1$. All paths from the root node to the bottom layer constitutes a vertex of the polytope of the non-integral solution space. Figure (b) shows the relevant solutions for this configuration. Clearly, Algorithm 5 computes most of the vertices more than once, and thus is very inefficient. Figure (c) shows the computation of the tree under the proposition that in every layer of it either a bin gets completely filled (depicted by a $\square$) or an item is completely distributed. In the first case, the bin is already full, which does not allow further packing (depicted by a $\square$), or the item has been packed yet further up in the tree (depicted by a $\diamondsuit$). These subtrees are cut off in (d). The grey nodes have been calculated by a lexicographically smaller ordering (a node further left in the tree). These nodes are shown in (d) connected to their lexicographically smaller equivalents by the dotted curves. (e) shows the whole tree traversal algorithm (cf. Algorithm 7).

The tree is traversed by depth fist search (DFS) from the left to the right. Lexicographically smaller nodes are automatically cut off during the search. The dotted branches are cut off due to the fact that either there is no capacity in the respective bin left (illustrated by a $\square$), or an item is completely distributed yet (illustrated by a $\diamondsuit$). The lexicographically smallest relevant solutions are depicted by the black nodes. They are saved in a solution set. The grey branches symbolize (partial) solutions that have been already generated by a branch further left in the tree, so they can be cut off safely without generating the specific subtrees. For example, the leftmost grey branch is the path $(2,2),(1,2)$. Clearly, due to the fact that in the father node of that path, bin 1 is completely filled yet, is does not matter, if bin 2 is filled with item 1 first, and then item 2, or vice versa. So the solutions are equivalent, and it is sufficient to save the leftmost of them.

Corollary 4.15. *The polytope of the non–integral solution space has at most* $\binom{MN-M+N}{M+N-1}$ *vertices.*

Proof. Any non–integral relaxation of an MBPP with M items and N bins can be written as a Linear Program with M equations and N inequalities:

$$
\begin{array}{lrcl}
(LP) & a_1x_{11} + a_1x_{12} + \ldots + a_1x_{1N} & = & d_1 \\
 & \vdots & & \vdots \\
 & a_Mx_{M1} + a_Mx_{M2} + \ldots + a_Mx_{MN} & = & d_M \\
 & a_1x_{11} + a_2x_{21} + \ldots + a_Mx_{M1} & \leq & c_1 \\
 & \vdots & & \vdots \\
 & a_1x_{1N} + a_2x_{2N} + \ldots + a_Mx_{MN} & \leq & c_N
\end{array}
$$

The dimension of the above system is $MN - M$. Using slack variables $s_1, \ldots, s_N$, the system can be transferred into a system of linear equations:

$$
\begin{array}{rcl}
a_1x_{11} + a_1x_{12} + \ldots + a_1x_{1N} & = & d_1 \\
\vdots & & \vdots \\
a_Mx_{M1} + a_Mx_{M2} + \ldots + a_Mx_{MN} & = & d_M \\
a_1x_{11} + a_2x_{21} + \ldots + a_Mx_{M1} + s_1 & = & c_1 \\
\vdots & & \vdots \\
a_1x_{1N} + a_2x_{2N} + \ldots + a_Mx_{MN} + s_N & = & c_N
\end{array}
$$

The dimension of the above system is at most $MN - M + N$. There are at most $M + N - 1$ non-zero variables per vertex. Thus the claim follows. $\square$

This is way too pessimistic. As we can see from Algorithm 7 in every step either i or j tuples are removed from the queue Q depending whether a bin gets filled or an item gets empty (or both). For clarity, we assume $M \equiv N$ for a first estimation. For example, let $M = N = 4$. In the first layer of the tree, we have $M \cdot N = 4^2 = 16$ nodes. Then, in the second layer, M or N (or both) is decreased by 1. Therefore, we have at most 12 successors for every node of the first layer. The maximal number of possible successors in layer 3 is then $3^2 = 9$, and so on. In the very last layer, every node has exactly one predecessor. In summary, there may be at most $16 \cdot 12 \cdot 9 \cdot 6 \cdot 4 \cdot 2 = 82944$ nodes in the tree. This observation is framed by the following lemma.

Lemma 4.16. *Let $M \equiv N$. Then the non-integral solution space has at most*

$$\prod_{n=2}^{M} (n^4 - n^3) \tag{4.48}$$

vertices.

Proof. The first layer of the compressed search tree consists of exactly $M \cdot N = M^2$ nodes. When descending in the tree by one layer, M or N (or both) is decreased by 1. In the next layer, there are then at most $M(N-1)$ direct successors of every node from layer 1. In the following layer, we may assume, there are at most $(M-1)(N-1)$ direct successors, because decreasing N would lead to less, namely $M(N-2)$ successors. Thus, alternatively decreasing M and N results in

$$MN \cdot M(N-1) \cdot (M-1)(N-1) \cdot (M-1)(N-2) \cdot (M-2)(N-2) \cdot \ \ldots \ \cdot 2 \cdot 1 \tag{4.49}$$

nodes in the bottom layer of the search tree at the utmost. Because of $M \equiv N$, the number of vertices in the non-integral solution space is at most

$$\prod_{n=2}^{M} (n^2 \cdot n(n-1)) = \prod_{n=2}^{M} (n^4 - n^3). \tag{4.50}$$

□

Theorem 4.17. *Let $M > N$. Then the non-integral solution space has at most*

$$\prod_{n=2}^{N} (n^4 - n^3) \cdot \prod_{n=N+1}^{M} (n \cdot N) \tag{4.51}$$

vertices.

Proof. The first layer of the search tree contains exactly $M \cdot N$ nodes. To obtain the maximal number of successors in every layer, M is decreased by 1 in every step descending in the tree as long as $M \geq N$. Therefore, there are at most

$$MN \cdot N(M-1) \cdot \ldots \cdot N(M-b+1) = \prod_{n=1}^{b} N(M-n+1) \tag{4.52}$$

nodes in the b-th layer. Choosing $b := M - N$ and applying Lemma 4.16, the claim follows.

□

Without loss of generality, we may regard the case $M > N$, because the case $N < M$ is proven analogously. In the following, we denote the set of vertices of the non-integral solution space by V.

Lemma 4.18. *The number of tree nodes v with pairwise different x_v is at most $|V| \cdot 2^{M+N-1}$*

Proof. For a tree node v, there is at least one vertex y of the non–integral solution space such that $x_v[i,j] = y[i,j]$ or $x_v[i,j] = 0$ for each $i \in \{1,\ldots,M\}$ and $j \in \{1,\ldots,N\}$. Corollary 4.10 implies that at most 2^{M+N-1} pairwise different vectors x_v may be related to a particular y in this way. □

Corollary 4.19. *The run time to compute all vertices of the non–integral solution space does not grow asymptotically faster than*

$$|V| \cdot 2^{M+N-1} \cdot (M+N-1)^2 .$$

Proof. Each check whether we can drop a tree node or not takes $O(M+N-1)$ time. By Lemma 4.18, there will be at most $O(|V| \cdot 2^{N+M-1})$ checks with a positive outcome in total. For each non-discarded tree node, there can be up to $O(M+N-1)$ checks for direct descendants with a negative outcome. □

4.4.9 Relaxation of Concave Constraints

Obviously, all concave constraints given in the original problem might be applied offhand if we set the decoupling relaxation from Section 4.4.3 aside. This is for example the case if we use the rounding procedure to k values from Section 2.1.3. If rounding values are decomposed into X and Δ parts, things are more difficult, because the given concave constraints have to be adapted to the new situation.

An easy relaxation of concave constraints is as follows: A concave constraint in the original problem translates to a relaxed concave constraint in the relaxed problem, whilst the constraint is imposed only on the X-part of a rounded item size $\tilde{a}_i$. Clearly, a concave constraint in the original problem cannot be imposed explicitly on the Δ-parts of rounded item sizes $\tilde{a}_i$, as items deliver various Δ-contributions to a rounding value, which are completely decoupled from the X-part. Concave constraints are imposed implicitly on the Δ-parts as well by reintroducing the X-Δ correspondence via additional constraints as it is done in Section 4.4.4. The technique used there might be applied to concave constraints as well: in case of unary constraints, the constraint imposed on the X-part is simply imposed to the corresponding number of Δ-parts, too. In case of constraints of higher arities, we might have to use upper and lower bounds for the Δ-parts.

This might be done as follows: consider for example identification constraints of the type

$$x_{ij} = 1, \text{ for all } i \in S_\ell \text{ and a } j \in \{1, \ldots, n\}, \tag{4.53}$$

which state that all items from the set S_ℓ have to be placed into bin j.

For any item $i \in S_\ell$ occurring in the constraint set(4.53), we regard the corresponding rounding value $\tilde{a}_i$. For each Δ_μ that is used for one of the rounding values $\tilde{a}_i := X_\mu + j \cdot \Delta_\mu$ to which an $i \in S_\ell$ is rounded down, we count the sum of all Δ_μ contributions occurring in S_ℓ by

$$\sigma_{\ell\mu} := \sum j \,\Big|\, i \in S_\ell \text{ is rounded down to } \tilde{a}_i = X_\mu + j \cdot \Delta_\mu. \tag{4.54}$$

Now, we are able to impose constraints on the number of Δ_μ-parts in bin j: at least $\sigma_{\ell\mu}$ of each Δ_μ have to be placed in bin j. These constraints are clearly concave.

Analogously, we can handle constraints that exclude items from bins. To do this, we have to count the sum $z_{\ell\mu}$ of the total Δ_μ contributions of all rounding values $\tilde{a}_i$, $i \in \{1, \ldots, m\}$ as it is done in equation (4.31). Now, we can impose constraints that at most $z_{\ell\mu} - \sigma_{\ell\mu}$ of each Δ_μ might be placed in bin j.

Many constraints might be relaxed automatically using the above technique. Moreover, it is possible to combine several original constraints in order to obtain tighter bounds on the number of Δ_μ values per bin. We are aware that neither every constraint from the original problem might have a reasonable relaxed analogon, nor all of these relaxations can be done automatically.

4.5 Infeasibility of Bin Packing Constraints and Concave Constraints

At first, we need theoretical insights about the relationship between polytopes and concave sets. Then, we are able to frame an algorithm which determines infeasibility of a Bin Packing Constraint with a set of arbitrary Concave Constraints by testing those constraints in pairs. In Section 4.5.2, we present an approach to check $n \leq dim(P)$ Concave Constraints simultaneously.

Observation 4.20. *Let $P \subseteq \mathbb{R}^n$ be a convex polytope and let $T \subseteq \mathbb{R}^n$ be a concave set. If $P \cap T \neq \emptyset$, then $P \cap T$ contains one of the vertices of P.*

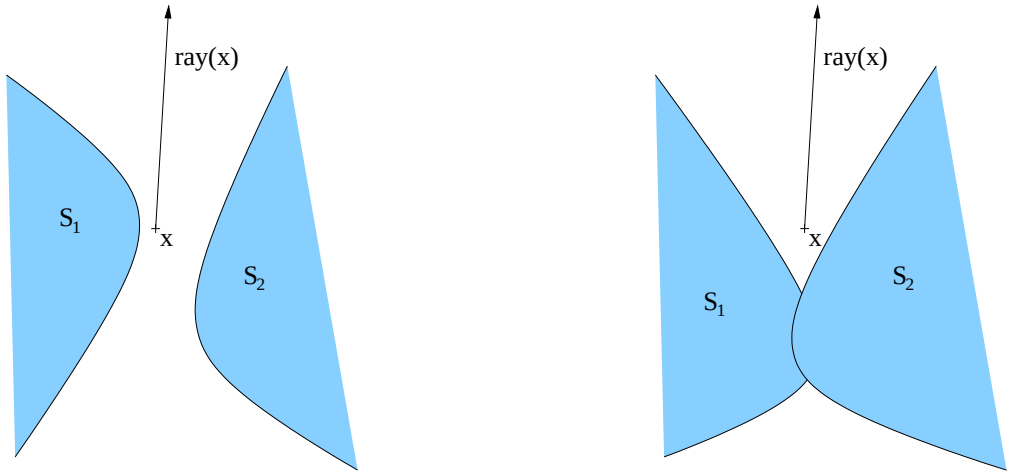


Figure 4.3: Configurations of convex sets $S_1, S_2 \subseteq \mathbb{R}^n$ and a vertex $\boldsymbol{x} \in \mathbb{R}^n \setminus (S_1 \cup S_2)$. On the left hand side with $S_1 \cap S_2 = \emptyset$, on the right hand side with $S_1 \cap S_2 \neq \emptyset$.

4.5.1 Determining infeasibility testing pairs of Concave Constraints

As usual, a *ray* at $\boldsymbol{x} \in \mathbb{R}^n$ is the set $\{\boldsymbol{x} + \lambda \cdot \boldsymbol{d} \,|\, \lambda \geq 0\}$ for some *direction* $\boldsymbol{d} \in \mathbb{R}^n$. The situation in the following lemma is illustrated in Figure 4.3 for the $\mathbb{R}^2$.

Lemma 4.21 (Auxiliary Lemma for Lemma 4.22)**.** *Let $n > 1$, $S_1, S_2 \subseteq \mathbb{R}^n$ be convex, and $\boldsymbol{x} \in \mathbb{R}^n \setminus (S_1 \cup S_2)$. Then there is an infinite ray of $\boldsymbol{x}$ that does not intersect S_1 or S_2.*

Proof. By induction on $n > 1$. For $n = 2$, the claim follows immediately from the following geometric observation: if a convex subset of $\mathbb{R}^n$ does not contain $\boldsymbol{x}$, it cannot span an angle of more than π as seen from $\boldsymbol{x}$; therefore, $S_1 \cup S_2$ cannot span the whole angle of 2π as seen from $\boldsymbol{x}$.

So consider $n > 2$. Suppose for a contradiction that every infinite ray of $\boldsymbol{x}$ intersects $S_1 \cup S_2$. Let $\boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathbb{R}^n$, and let $\lambda_1, \lambda_2 \geq 0$ be maximal such that $\boldsymbol{x} + \lambda \cdot \boldsymbol{y}_1 \notin S_1 \cup S_2$ for all $\lambda \in [0, \lambda_1)$ and $\boldsymbol{x} + \lambda \cdot \boldsymbol{y}_2 \notin S_1 \cup S_2$ for all $\lambda \in [0, \lambda_2)$. Since S_1 and S_2 are convex, $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ can be chosen such that $\boldsymbol{x} + \lambda_1 \cdot \boldsymbol{y}_1 \in \partial S_1$ and $\boldsymbol{x} + \lambda_2 \cdot \boldsymbol{y}_2 \in \partial S_2$. Clearly, there is a tangent hyperplane H_1 for ∂S_1 at $\boldsymbol{x} + \lambda_1 \cdot \boldsymbol{y}_1$, and a tangent hyperplane H_2 for ∂S_2 at $\boldsymbol{x} + \lambda_2 \cdot \boldsymbol{y}_2$. Then S_1 is on one side of H_1 (incl. H_1) and $\boldsymbol{x}$ is on the other side of H_1 (incl. H_1). Analogously, S_2 is on one side of H_2 (incl. H_2), and $\boldsymbol{x}$ is on the other side of H_2 (incl. H_2).

If $\boldsymbol{x} \in H_1$ (or, analogously, $\boldsymbol{x} \in H_2$), we may apply the induction hypothesis to conclude that there is an infinite ray of $\boldsymbol{x}$ in H_1 which does not intersect $S_1 \cup S_2$, because $S_1 \cap H_1$ and $S_2 \cap H_1$ are convex. On the other hand, if $\boldsymbol{x} \notin H_1 \cup H_2$, we may analogously apply the induction hypothesis to the plane parallel to H_1 (or H_2) that contains $\boldsymbol{x}$. □

Lemma 4.22. *Let $P \subseteq \mathbb{R}^n$ be a convex polytope and let T_1 and T_2 be two concave sets such that $P \cap T_1 \cap T_2 \neq \emptyset$. Then, there are vertices $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ of P such that $\boldsymbol{x}_1 \in T_1$, $\boldsymbol{x}_2 \in T_2$, and either it is $\boldsymbol{x}_1 = \boldsymbol{x}_2$, or $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are neighbored vertices of P.*

Proof. By induction on the dimension $d \geq 0$ of the polytope P. For $d = 0$ and $d = 1$, nothing is to show. So suppose $d > 1$.

Lemma 4.21 shows that, for any two convex sets and a vertex $\boldsymbol{x}$ that is not part of any of these sets, there is an infinite ray that does not intersect any of these sets. Therefore, for the concave

complement $T_1 \cap T_2$, we know that each element of $T_1 \cap T_2$ has a ray completely inside $T_1 \cap T_2$. Let $\boldsymbol{x} \in P \cap T_1 \cap T_2$, and let $\boldsymbol{y}$ be a ray for $\boldsymbol{x}$ in $T_1 \cap T_2$. Then there is $\lambda_0 \geq 0$ such that $\boldsymbol{x} + \lambda \cdot \boldsymbol{y} \in P$ for all $\lambda \in [0, \lambda_0]$ and $\boldsymbol{x} + \lambda \cdot \boldsymbol{y} \notin P$ for all $\lambda > \lambda_0$. Let $\boldsymbol{x}' := \boldsymbol{x} + \lambda_0 \cdot \boldsymbol{y}$. Then $\boldsymbol{x}' \in \partial P$. Let P' denote a face of P that contains $\boldsymbol{x}'$. In particular, P' is a convex polytope of dimension at most $d - 1$.
As $\boldsymbol{x}' \in P' \cap T_1 \cap T_2$, we can apply the induction hypothesis. Hence, there are vertices $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ of P' such that $\boldsymbol{x}_1 \in T_1$ and $\boldsymbol{x}_2 \in T_2$, and either it is $\boldsymbol{x}_1 = \boldsymbol{x}_2$, or $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are neighbored in P'. As all vertices of P' are also vertices of P and two neighbored vertices of P' are also neighbored in P, we have proven the induction step, and thus Lemma 4.22. □

Lemma 4.22 gives rise to a strategy for evaluating a Bin Packing Constraint and an arbitrary number of additional Concave Constraints simultaneously. We summarize this approach in the following algorithm. Note that this procedure can be easily and efficiently applied incrementally whenever new concave constraints are added to the constraint store. In fact, at the end of the preprocessing, we may use all concave constraints in the original constraint store to compute a certificate that the instance is infeasible. Afterwards, during the search, adding a new concave constraint amounts to checking the existence of vertices as guaranteed by Lemma 4.22 for all pairs consisting of this new constraint and other constraints. Whenever this test fails for any pair of constraints, we have a certificate that we may safely deliver "*infeasible*". Figure 4.4 shows different configurations if the problem is feasible.

Algorithm 8 `DetermineInfeasibility`

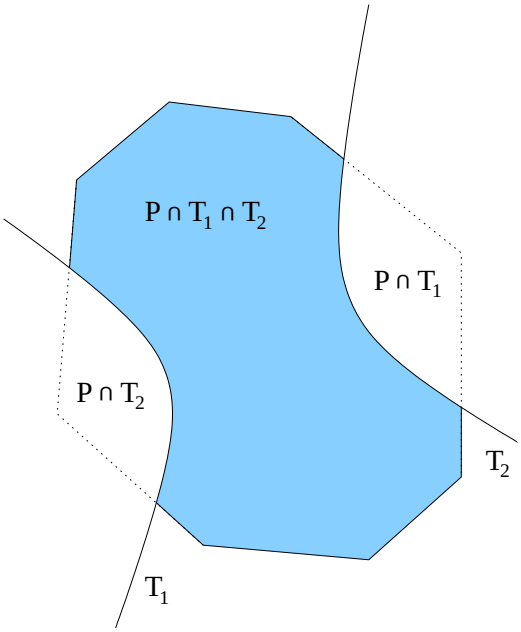
compute the vertices of polytope P of the non-integral relaxation of the Bin Packing Constraint using Algorithm 7
maintain a constraint store $\mathcal{C}$
relax all given concave constraints (cf.Sec 4.4.9) and add them to $\mathcal{C}$
add X-Δ-correspondence as additional concave constraints (cf.Sec. 4.4.4) to $\mathcal{C}$
add relaxed integrality constraints (cf.Sec. 4.4.5) to $\mathcal{C}$
foreach concave constraint in $\mathcal{C}$ **do**
    find the set of all vertices of P satisfying this constraint
foreach pair of constraints in $\mathcal{C}$ **do**
    determine whether their corresponding sets of vertices overlap or
    are at least neighbored via one pair of neighbored vertices of P
if this test fails for at least one pair of concave constraint **then**
    return "*infeasible*"
else
    return "*potentially feasible*"

4.5.2 Determining infeasibility testing n Concave Constraints

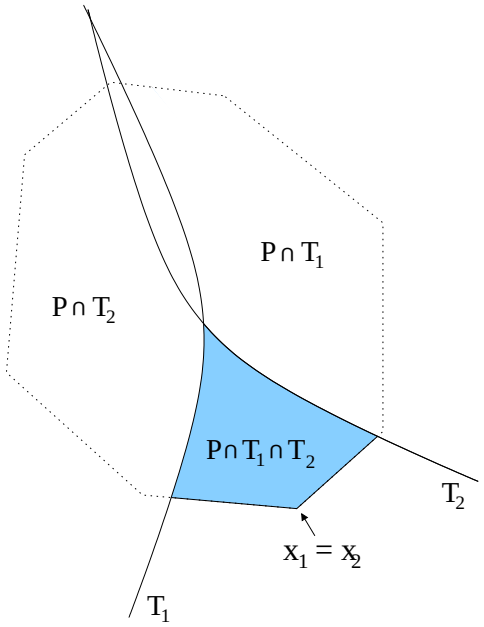
Generally, we may check more than 2 Concave Constraints at the same time. In order to achieve this, we will have to take the following theoretical considerations into account.

Lemma 4.23. *Let $S_1, \ldots, S_m \subseteq \mathbb{R}^n$ convex sets, $m \leq n$ and $\boldsymbol{x} \in \mathbb{R}^n \setminus (S_1 \cup \ldots \cup S_m)$. Then there is an infinite ray of $\boldsymbol{x}$ that does not intersect $S_1, \ldots, S_m$.*

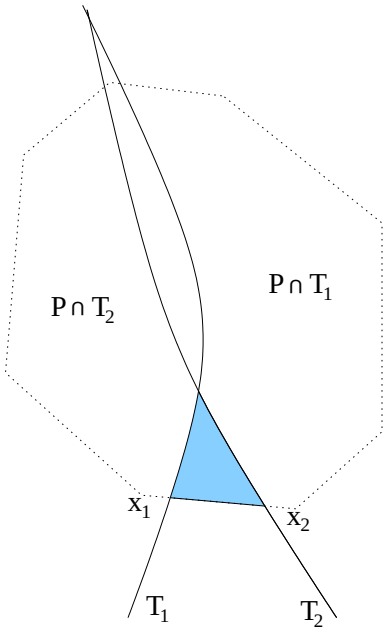
Proof. For a contradiction, suppose that every infinite ray of $\boldsymbol{x}$ intersects $S_1 \cup \ldots \cup S_m$. Choose $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m \in \mathbb{R}^n$ and let $\lambda_1, \ldots, \lambda_m$ be maximal such that $\boldsymbol{x} + \lambda \boldsymbol{y}_j \notin S_1 \cup \ldots \cup S_m$ for $\lambda \in [0, \lambda_j)$ $\forall j \in \{1, \ldots, m\}$. Since $S_1, \ldots, S_m$ are all convex, $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m$ may be chosen such that $\boldsymbol{x} + \lambda_j \boldsymbol{y}_j \in \partial S_j$ $\forall j \in$



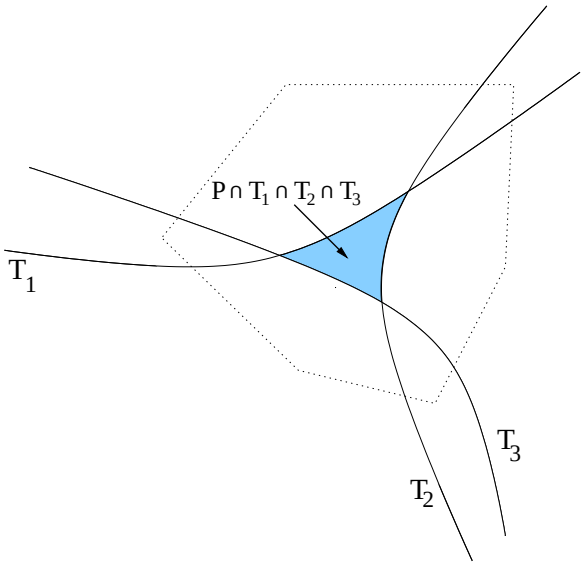
(a) cutset yielding a large feasible region



(b) cutset with one common vertex of P



(c) cutset with two neighbored vertices



(d) cutset with three concave constraints

Figure 4.4: Different constellations of cut-sets of the polytope P and the concave constraints T_1 and T_2. The colored region may contain potentially feasible integral solutions. **(a)** In many cases we may have quite weak concave constraints that yield a large feasible region. **(b)** If this region is smaller, there is either one common vertex in $P \cap T_1 \cap T_2$ or **(c)** there are vertices $\boldsymbol{x}_1 \in T_1$, $\boldsymbol{x}_2 \in T_2$ that are neighbored in ∂P. In all cases we are able to find vertices as stated in Lemma 4.22. Only if there are no such vertices, the feasible region is empty and we can say for sure that the instance is infeasible. The concave constraints have to be checked pairwise for the reason we work with the convex hull ∂P. Otherwise we cannot infer from the lack of (neighbored) vertices of P to the non-existence of a feasible region, because a cut-set consisting of more constraints than the dimension of P may yield a feasible region consisting of interior points in $P \backslash \partial P$ (see example **(d)**).

$\{1, \ldots, m\}$. Therefore, there is a tangent half-space H_j for each ∂S_j at $\boldsymbol{x} + \lambda_j \boldsymbol{y}_j$. For every H_j, S_j is on one side of the hyperplane ∂H_j induced by H_j, and $\boldsymbol{x}$ is on the other side of this hyperplane. Furthermore is $\boldsymbol{x} \in H_j$. As $H_1 \cap \ldots \cap H_m$ describes an unbounded polyhedron in $\mathbb{R}^n$, there is an unbounded direction. Thus, there is an infinite ray of $\boldsymbol{x}$ that does not intersect $S_1 \cup \ldots \cup S_m$. □

Lemma 4.24. *Let $P \subseteq \mathbb{R}^n$ be a convex polytope and $T_1, \ldots, T_m,\ m \leq n$ concave sets such that $P \cap T_1 \cap \ldots \cap T_m \neq \emptyset$. Then there are vertices $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ on a face of P such that $\boldsymbol{x}_i \in T_i\ \forall i \in \{1, \ldots, m\}$.*

Proof. From Lemma 4.23 we know that each element of $T_1 \cap \ldots \cap T_m$ has a ray completely inside $T_1 \cap \ldots \cap T_m$. Let $\boldsymbol{x} \in P \cap T_1 \cap \ldots \cap T_m$, and let $\boldsymbol{y}$ be a ray for $\boldsymbol{x}$ in $T_1 \cap \ldots \cap T_m$. Then there is $\lambda_0 \geq 0$ such that $\boldsymbol{x} + \lambda \cdot \boldsymbol{y} \in P$ for all $\lambda \in [0, \lambda_0]$ and $\boldsymbol{x} + \lambda \cdot \boldsymbol{y} \notin P$ for all $\lambda > \lambda_0$. Let $\boldsymbol{x}' := \boldsymbol{x} + \lambda_0 \cdot \boldsymbol{y}$. Then $\boldsymbol{x}' \in \partial P$. Let P' denote a face of P that contains $\boldsymbol{x}'$. In particular, P' is a convex polytope of dimension at most $d - 1$. Since $\boldsymbol{x}' \in P' \cap T_1 \cap \ldots \cap T_m$ and $T_i \cap P' \neq \emptyset\ \forall i \in \{1, \ldots, m\}$, there is a vertex $\boldsymbol{x}_i \in P'$ for every concave set T_i. □

Note, that the number of concave sets may be at most n. When we choose $n + 1$ convex sets, the resulting polyhedron might be an n-simplex, which is clearly bounded. This is the reason we can check only $n \leq dim(P)$ Concave Constraints at the same time. Otherwise, if $n > dim(P)$,the region defined by $n + 1$ Concave Constraints may contain integral points although there is no face of P that satisfies Lemma 4.24. This situation is illustrated in Figure 4.4d.

4.6 Discussion of Accuracy and Efficiency

Clearly, if the original instance (with the integrality requirement on all values x_{ij}) is feasible, this procedure will deliver '*potentially feasible*'. Recall that this was the basic requirement. However, it may happen that the procedure delivers '*potentially feasible*' although the original instance is infeasible. This is of course due to the relaxations. Therefore, we will give a short informal discussion of the impact of the relaxation here. Recall that we have applied the following relaxations to the original problem:

(i) replacing the original item sizes a_i by rounding values $\tilde{a}_i$ (cf. Section 4.4.2),

(ii) decoupling the X values from the Δ values (cf. Section 4.4.3),

(iii) relaxing the problem in a non–integral fashion (cf. Section 4.4.6), and

(iv) the way in which the concave constraints are relaxed and tested (cf. Sections 4.4.9 and 4.5).

The rounding error of an appropriate rounding technique from Chapter 2 can be reduced to minimality. Regardless whether rounding to K arbitrary values or Adaptive Rounding is used, reasonable bounds on the error are proven, and comprehensive computational experiments show that the effect of this relaxation cannot be too large. Clearly, the larger the number of rounding values, the smaller the effect of this relaxation will be.

When Adaptive Rounding is used, the relaxation in (ii) might typically not be too strong because the dropped X–Δ–correspondence is reintroduced to a significant extent in form of additional concave constraints (cf. Section 4.4.4). Heuristically, the larger the number of items for each size is, the smaller the effect of this relaxation will be. The number of items is particularly large if the number of sequences and the sizes of the sequences are large. Therefore, we obtain exactly the same tradeoff as in (i). Clearly, if we use geometric rounding to K values, this relaxation is skipped.

Regarding the relaxation in (iii), it can be seen that the number of non–integral values of a vertex is at most $2N - 2$. So, if at least one vertex is delivered by Algorithm 5 for some choice of "$\prec$", the

original Bin Packing Constraint is feasible for some capacities c'_i instead of c_i, where $c'_j \geq c_j$ for all $j \in \{1, \ldots, N\}$ and

$$\sum_{j=1}^{N} \left(c'_j - c_j\right) \leq (2N-2) \cdot \max\left\{a_i \,\middle|\, i \in \{1, \ldots, M\}\right\}.$$

So, whenever the item sizes are small compared to the bin capacities, the impact of the non–integral relaxation is not too large. Moreover, integrality is imposed in the form of additional concave constraints (cf. Section 4.4.5).

Finally, the relaxation of concave constraints has certainly an impact on the efficiency of infeasibility detection. This is mainly due to the dropping of X–Δ–correspondences. However, as the correspondences are partly reestablished by additional concave constraints in Section 4.4.4, the concave constraints are implicitly involved in this process, too. As we have seen in Section 4.4.9, there are relaxed equivalents to many of the elementary concave constraints, although not all of them can be derived automatically.

Considering the performance of the presented framework, the worst-case bounds in Section 4.4.8 might be quite terrifying. Nevertheless, we must not forget that this is nothing new in the context of Constraint Programming: the realization of an entire constraint propagation is usually $\mathcal{NP}$-complete, in particular if global constraints are involved that are deduced from $\mathcal{NP}$-hard problems such as BIN PACKING. Therefore, a total enumeration generally results in an exponential runtime. This is one of the main reasons why constraint propagation is usually implemented incompletely in constraint solvers. However, there is always hope to detect infeasibility at an early stage in the framework. This might be achieved by testing 'strong' concave constraints that are satisfied by only a small subset of vertices first.

4.7 Summary

In this chapter, we investigated the applicability and evaluation of the Bin Packing Constraint together with a set of Concave Constraints, a paradigm that mainly arose from Constraint Programming. We introduced a concept that can be used to model various optimization and decision problems and goes far beyond the capabilities of modeling problems by using only linear constraints. We have shown that our framework is applicable to a broad class of problems from Operations Research and Combinatorial Optimization. It is therefore of particular practical interest. We put special emphasis on modeling nonlinear constraints. Finally, we presented an algorithmic approach to evaluate Bin Packing Constraint jointly with a set of Concave Constraints in order to detect infeasibility of a given constraint system. Within this scope, we developed techniques to efficiently enumerate the vertices of a polytope of the non–integral relaxation of the problem and gave theoretical consideration on how to test up to n Concave Constraints simultaneously. Concludingly, we presented worst-case bounds on the complexity of the framework.

5 Solving a highly–constrained Scheduling Problem from PCB assembly line optimization

The first 90% of the code accounts for
the first 90% of the development time.
The remaining 10% of the code accounts for
the other 90% of the development time.

— Tom Cargill (Bell Labs) BENTLEY (1985)

5.1 Introduction

Throughput optimization of assembly lines has attracted a lot of attention over the past years. Due to the rapid expansion of production volumes in electronics manufacturing, the assembly of printed circuit boards (PCBs) must be run more efficiently than ever. At the same time, the range of product variants has widened, and production cycles are steadily narrowing. Thus, a high mix of product series of low and medium volumes has to be produced at very tight schedules.

The production of several board types is done in batch processing. Usually, board types of one distinct branch of electronics, e.g. automotive parts or mobile phones, which differ only slightly with respect to their sets of utilized component types are combined to one batch job. For this reason, a typical batch job implies a machine setup that varies only slightly from board type to board type. Some PCBs might have need for special component types to be set up onto the assembly line. For all of the above reasons, quick partial setup changes are practically highly relevant.

5.1.1 Partly Combined Processing

These quick partial setup changes can be realized by *partly-combined processing*: Consider an assembly line that consists of multiple *placement modules*, or *modules* for short, in a row; these are robots that pick components and place them on the PCBs that pass sequentially through the line. Each of these modules holds a distinct set of component types, which is called its *setup*. During the processing of a batch job, most placement modules keep a fixed setup. However, selected modules feature a *variable setup*, which is realized by one or more *pluggable trolleys*. These trolleys might be hot swapped several times during a batch job in order to allow several partial setup changes. Spare trolleys for forthcoming PCB types in the batch may be set up offline, so the processing needs not to be interrupted for the total time a setup change would consume, but only shortly for the exchange of trolleys.

Moreover, due to the fact that frequently used component types can be assigned to more than one module, partly-combined processing may boost the degree of parallelism, and therefore, leads to higher throughput rates. Idle times originating from setup changes are reduced to a minimum

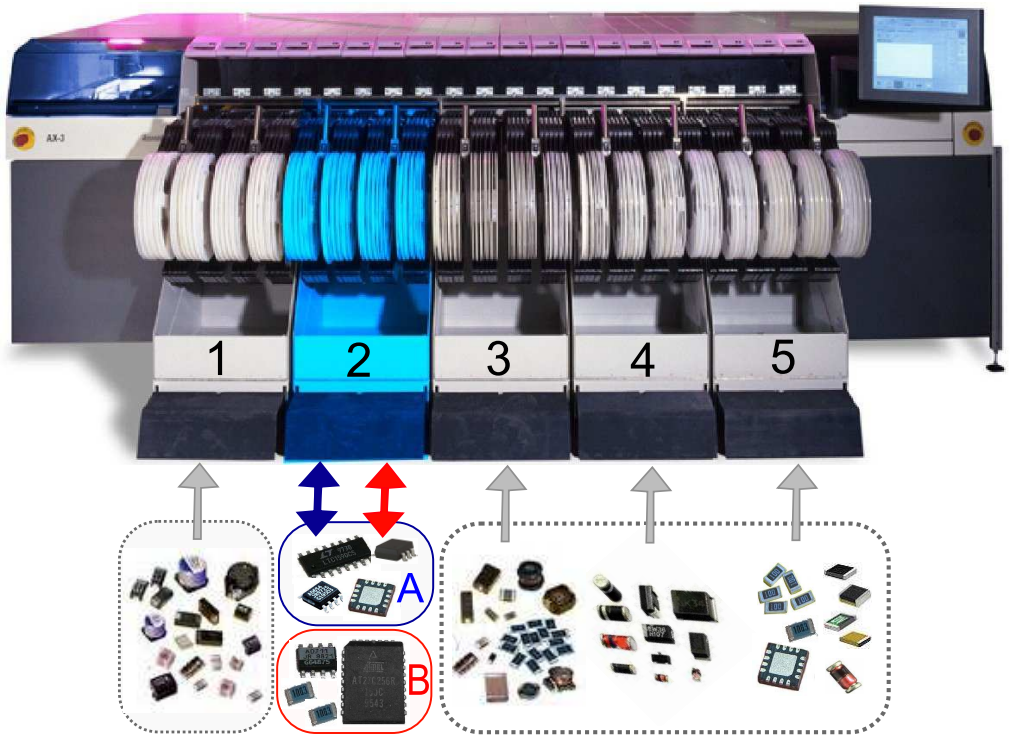


Figure 5.1: Partly-combined processing realized by exchangeable trolleys: trolley number 2 may be changed between the processing of two distinct PCB types. This allows a partial change in the machine setup whereas the larger part of the setup remains fixed: Before the exchange, components from the set A are to be mounted, afterwards the setup of trolley 2 consists of components from set B.

due to the fact that the setup changes are done offline during processing. Figure 5.1 shows such a configuration. During the processing of a batch of multiple PCB types, trolley number 2 may be hot-swapped several times.

5.1.2 Previous Work

There is extensive work on the optimization of PCB assembly lines, so we will only give an overview of publications that are relevant for our purposes in some sense.

Most approaches in the literature are addressing the Job Grouping and Machine Setup Problem apart from each other. The heuristic approach from MAGAZINE ET AL. (2002) is an exception as it uses an integrated model for clustering and machine setup. However, this work considers full tear-down setups only, and not the partly-combined case.

For the problem of partitioning the PCBs into groups, several strategies can be found in the literature. The approaches range from very generic considerations, like similarity measures for cluster merging (SMED ET AL., 2000), to more and more specific approaches for distinct machine types and scenarios (KNUUTILA ET AL., 2004, 2005). The majority of job grouping approaches uses *agglomerative clustering* (SALONEN ET AL., 2000, 2003; STEINBACH ET AL., 2000), which simply merges groups iteratively according to a certain similarity measure. The whole scheduling problem is highly constrained, and agglomerative clustering does not take any of these constraints into account. Therefore, it is not a priori clear whether a computed grouping admits a feasible setup. This is a big issue as

with every merge operation the existence of a feasible setup has to be checked in order to guarantee a feasible grouping.

Moreover, the majority of approaches in the literature aims at the minimization of the number of groups. We prefer a different objective that is, to our opinion, much more realistic: minimize the makespan provided that the number of trolley changes is not too large. For example, ELLIS ET AL. (2003) present a model that minimizes the total production time incorporating the tradeoff between production and setup times. This can easily be achieved by incorporating the setup changes into the objective function.

Our objective is indeed much more adequate in the case of partly-combined processing: Setup times are roughly negligible if spare trolleys are set up offline. Naturally, the number of setup changes should be kept at a reasonable scale, due to the fact that each trolley exchange requires expensive manpower, and may be fault-prone as well.

In contrast to the Setup Problem in case of only one single board type, there is not much literature about the Setup Problem for partly-combined production scenarios. The articles of AMMONS ET AL. (1997), CRAMA ET AL. (2002), and JOHNSSON & SMED (2001) give a general classification of setup strategies in PCB assembly optimization. Partly-combined setups are mentioned there as well. LEON & PETERS (1996) give a conceptual formulation of the underlying problems and primarily illuminate the advantages arising from a partly-combined processing strategy. LOFGREN & MCGINNIS (1986) consider *partial setups* on a component sequencer machine on which component-feeder assignments are not an issue (as opposed to our case). However, these assignments are an essential feature of most current component mounters, and thus are crucial for the feasibility of setups. Partial setups are also considered in a tactical planning model for PCB assembly using a column generation approach (BALAKRISHNAN & VANDERBECK, 1999). However, that scenario is quite different from ours and the test jobs are much smaller than the ones we will consider in this work.

5.1.3 Our Contribution

Optimizing the automated manufacturing of PCBs is a highly complex task. Strategies in the literature are often highly machine specific, and therefore are complicated or even impossible to adapt to the characteristics of other types of machines. Monolithic models, for example integer linear programs (ILP), grow vastly and are not even approximately tractable with state-of-the-art solvers within reasonable time (see our model in Section 5.2.3). Due to this insight, which is also supported by our computational experiments, we consider a series of subproblems and solve them consecutively. The major stages in this waterfall approach are (1) the *Job Grouping Problem*, (2) the *Machine Setup Problem*, and (3) the distribution and sequencing of placements (CRAMA ET AL., 2002). Once a grouping and a setup have been computed, in practice, we are able to solve the third problem nearly to optimality in polynomial time using a maximum flow algorithm (MÜLLER-HANNEMANN ET AL., 2006). Our empirical studies have shown that the Job Grouping and the Setup Problem are crucial for the complete framework, so we will have a closer look at these two here. The combined problem is shown to be strongly $\mathcal{NP}$–hard in Section 5.3.

In this work, we will present a novel approach to Job Grouping. In contrast to other approaches which merge groups exclusively based on a similarity measure, our technique incorporates essential machine constraints in order to obtain feasible groupings. For each component type, the decision whether it is assigned to a module with a static setup or to a module with an exchangeable setup is made during the grouping as well. Eventually, we aim at a balancing of workload among the individual modules, so duplication of frequently used component types to more than one module is essential. In order to support load balancing at an early stage in our approach, the component-to-

module assignments are balanced between the static and the exchangeable parts of the setup already during the grouping process.

Moreover, we present an approach to setup computation consisting of 2 stages: Given a feasible grouping, we first compute an initial feasible partly combined setup. In order to exploit the high parallelism of the machine and to provide load balancing between individual modules, these initial setups are further improved by duplicating frequently used component types. The above mentioned preference, whether component types are assigned to modules with either a static or a dynamic setup, is used in the setup algorithms. This two-stage approach delivers feasible setups even if the batch jobs are very tight due to the large number of distinct boards.

We are going to address the Job Grouping and the Setup Problem in a sufficiently generic way, which allows an easy adaption to specific needs of a wide variety of scenarios and machine types.

The suggested approach has been implemented and tested on several difficult real-world instances from automotive and cell-phone manufactures, with batch jobs of up to 42 different board types and with as many as about 26,000 mounting tasks. In a cooperation with Philips Assembléon, B.V., Eindhoven, The Netherlands, we have specialized our implementation to their fast component mounter series AX and integrated all specific constraints of this machine. Our computational results show that our approach significantly outperforms a previous approach both by quality and computational speed. Even for the largest instances we obtain good quality solutions within a few minutes, making our scheduling approach immediately applicable in practice.

Overview

In Section 5.2, we will give a detailed formal problem description of the specific scheduling problem, and the Job Grouping and Setup Problem in particular. Furthermore, we state a comprehensive ILP model of the whole problem with special attention to the grouping and setup constraints. Section 5.3 proves strong $\mathcal{NP}$–hardness of this problem. In Section 5.4, we identify criteria for successfully grouping PCBs with particular regard to feasibility and workload balancing issues. Based on these considerations, we develop an algorithm for the Job Grouping problem in the partly combined case. We propose a new techniques to compute feasible setups. In Section 5.5, we present computational experiments using several real-world production problems and compare our framework to another approach using Agglomerative Clustering and a setup procedure similar to the approach from AMMONS ET AL. (1997).

5.2 Formal Problem Description

We will consider the Setup Problem in Section 5.2.4 and the Job Grouping Problem in Section 5.2.5. In the following, we will draw an outline of the underlying scheduling problem.

5.2.1 Framework of the Scheduling Problem

We consider an assembly line $\mathcal{M}$ consisting of n, not necessarily identical, *placement modules* or *modules* for short, $\mathcal{PM} := \{pm_1, \ldots, pm_n\}$. These are robots, which pick components from a *component feeder* and mount them on a printed circuit board. Modules are characterized by the *toolbits* they hold, and therewith the set of *component types* they are able to mount. Furthermore, modules have a *feeder capacity restriction* limiting the number of different component types they are served with. This is due to the alignment of the modules next to each other along the line (cf. Figure 5.1).

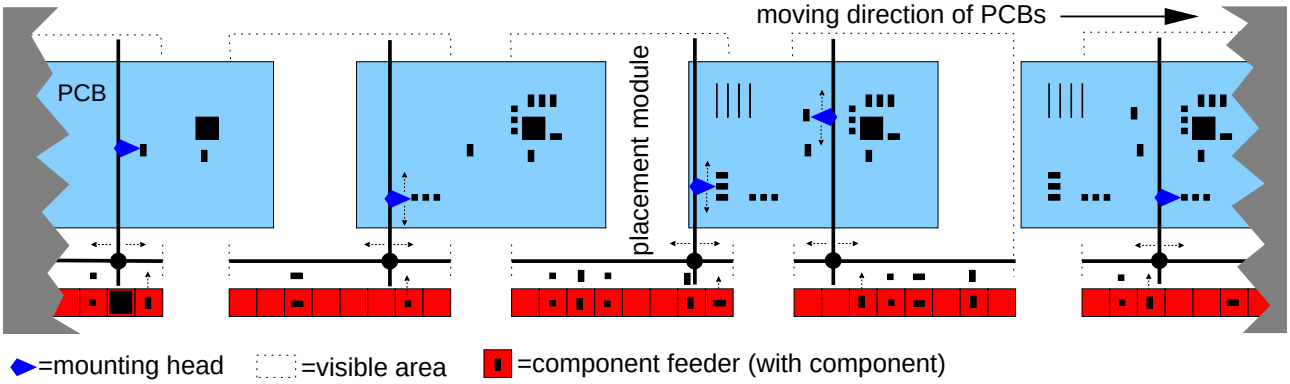


Figure 5.2: Overview of the machine model: PCBs are moved stepwise through the assembly line. The placement modules pick components from the component feeders and place them on the PCBs. As this is done in parallel, multiple modules might place component types on the same board.

A batch job $\mathcal{W}$ consisting of PCB types $p_1, \ldots, p_k$ has to be carried out on $\mathcal{M}$. On each PCB type p_ℓ, a set of *tasks* $J(p_\ell)$ has to be executed. Each task $t \in J(p_\ell)$ places a certain component type $ct(t)$ to a specific location on the PCB. We denote by $\mathcal{J}$ the set of all tasks over all board types, and by $\mathcal{CT}$ the set of component types used by $\mathcal{J}$. In order to place a component type, a certain *toolbit type* on the module is necessary. Modules may exchange toolbits during operation in order to mount a larger set of component types, but the set of toolbits is always limited due to space restrictions. Therefore, not every module is capable of mounting a specific component type.

We are given an *index scheme* S_p, which comprises the periodical forward movement of a board of type p in the line. An index scheme is provided for each board type separately. A scheme S_p contains a number of *index steps* $s_1, \ldots, s_z$. Each of these index steps defines an offset in the direction of the assembly line. The boards are conveyed stepwise. A board, that is at a certain position in the line, will be at the same position as the preceding board once the whole scheme has been executed. So, for each module and each index step s_ℓ, we know exactly which parts of the boards are *visible* to a module (cf. Figure 5.2).

In our specific application, which motivated this work, several component feeders are combined on one *trolley*, which is pluggable to the machine (cf. Figure 5.1). In practice, one or more of these trolleys might be set up offline, and then exchanged on the fly during processing. To support other machine types as well, we will choose a more abstract point of view here: We split the set of placements modules $\mathcal{M}$ into a set $\mathcal{SM}$ of modules, whose sets of component types are not exchangeable during a batch, i.e. they are said to have a *static setup*, and the set $\mathcal{DM}$ of modules, whose sets of component types may be exchanged between different board types. Analogously, their setup is called *dynamic*. The same holds for the sets of toolbits. During a batch job, ℓ changes in the setup on $\mathcal{DM}$ imply a partition $\mathcal{G}$ on the board types into $\ell + 1$ groups. This means, each two PCBs in the same group share the same component and toolbit setup on the entire machine $\mathcal{M} = \mathcal{DM} \cup \mathcal{SM}$ during the processing of this very group.

Eventually, note that there is no specific ordering on the sequence of board types in the input. This is an essential difference to the definition of a partial setup strategy given by JOHNSSON & SMED (2001). As the placement modules featuring dynamic setups are predefined and their positions are always the same in the line, the processing sequence of the individual groups can be permuted arbitrarily as well as the processing sequence within a group.

5.2.2 Relation to Bin Packing

Multibin Packing Problems

The general concept of *Multibin Packing* extends several well-known problems such as machine scheduling, project planning and load balancing. Unlike in BIN PACKING, where objects have to be packed as a whole, in Multibin Packing, objects are made of several identical parts that might be distributed over different bins. The number of parts may vary from object to object. This concept was first introduced by LEMAIRE ET AL. (2003a,b, 2006) and can be seen as a relaxation to the original Bin Packing Problem. Depending on the application, there might be some additional constraints concerning the partition and distribution of objects: for example, objects might only be divided into parts of identical size, or parts must always be packed into different bins.

The Multibin Packing Problem includes several well-known problems such as bin packing, multiprocessor scheduling, partitioning, or the ring loading problem (SCHRIJVER ET AL., 1998). They have many applications in network design, project planning and scheduling problems.

The Scheduling Problem in terms of Multibin Packing

In our case the bins $\{1,\ldots,n\}$ are formed by the placement modules. The (variable) size of a bin j is the module's workload wl_j. Assuming that all mounting tasks have the same duration, namely unit length, the module's workload is the sum of all mounting tasks assigned to j, that is $wl_j := \sum_{i=1}^{m} x_{ij}$. An object i consists of the set of all mounting tasks using the same component type. So there are $m := |\mathcal{CT}|$ objects of integral sizes $p_i := |\bigcup t|, t \in J$ with $ct(t) = i$ for $i \in \{1,\ldots,m\}$. Each of them may be split into at most $\max\{p_i, \ell_{\max}\}$ different parts and may be assigned to at most n different modules. Our objective is to distribute all objects over the bins while minimizing the maximal workload (cf. (5.1)). By x_{ij} we denote the integral part of object i that is assigned to bin j. By the parameter w_{pm} we denote the maximal capacity of a placement module, that is the maximum number of different component types it is able to handle. Now, we are able to formulate the problem as an integer linear programming model:

$$\textit{minimize } \max_j \{wl_j := \sum_{i=1}^{m} x_{ij}\} \tag{5.1}$$

$$\begin{aligned}
\text{s.t.} \quad \sum_{j=1}^{n} x_{ij} &= p_i, \text{ for all } i \in \{1,\ldots,m\}, && (5.2)\\
y_{ij} \cdot M &\geq x_{ij}, \text{ for all } i \in \{1,\ldots,m\},\ j \in \{1,\ldots,n\}, && (5.3)\\
y_{ij} &\leq x_{ij}, \text{ for all } i \in \{1,\ldots,m\},\ j \in \{1,\ldots,n\}, && (5.4)\\
\sum_{i=1}^{m} y_{ij} &\leq w_{pm} \text{ for all } j \in \{1,\ldots,n\}, && (5.5)\\
\sum_{j=1}^{n} y_{ij} &\leq \ell_{\max} \text{ for all } i \in \{1,\ldots,m\}, && (5.6)\\
x_{ij} &\in \mathbb{N}_0, && (5.7)\\
y_{ij} &\in \{0,1\}. && (5.8)
\end{aligned}$$

Equation (5.2) ensures all mounting tasks are executed exactly once. The big-M constants in (5.3) and (5.4) may be chosen as $M := p_{\max} := \max\{p_i\}$. These two inequalities force y_{ij} to be 1 iff $x_{ij} \neq 0$, and 0 iff $x_{ij} \equiv 0$. This helps counting the number of component types per placement module for the feeder capacity bound w_{pm} in equation (5.5), and the maximum number of modules $\ell_{\max}$ one specific component might be assigned to (5.6). Aside from the feeder capacity constraint (5.5), the above model corresponds to the case $Bm|any_j|H_{\max}$ in standard scheduling notation (LEMAIRE ET AL., 2003a).

Extension: Partly-Combined Setups

In the previous formulation, we have disregarded the partial setup strategy consciously in order to obtain a more general model. The model can be easily extended to the partly-combined case: we add additional virtual workstations, namely exactly the number of workstations served by feeders located on a dynamic trolley multiplied by the number of used groups g. Then, the set $\mathcal{DM}$ of modules with a dynamic setup manifolds to g sets $\mathcal{DM}_\ell$, one for each group ℓ. Treating the setup problem in this manner, we have to additionally ensure, that the components required by the boards in a group are present in the current setup. By $p_{i\ell}$ we denote the number of mounting tasks in group ℓ that use component type i. As the grouping is known a priori, this is simply the sum of mounting tasks using the same component types of all boards in this group. To ensure that all mounting tasks are processable, we have to ensure that all necessary component types for the processed board i are assigned in the specific group ℓ. This is done by adding the constraints

$$\sum_{j \in \mathcal{SM}} x_{ij} + \sum_{j \in \mathcal{DM}_\ell} x_{ij} \quad \geq \quad p_{i\ell}, \quad \text{for all } i \in \{1, \ldots, m\},\ \ell \in \mathcal{G}. \tag{5.9}$$

Clearly, the objective function (5.1) must be changed to fit into the partly combined concept. This may be done by additionally indexing the x_{ij}-variables by the groups and minimizing the maximum workload in each group.

Additional Constraints

One criterion for the use of dynamic trolleys is of course the wide variety of component types occurring during the setup of a batch job. An even more crucial point for the throughput of the machine yet is the duplication ratio, that is the multiplicity of component-to-module assignments. Certainly, the throughput increases if component types, that are to be mounted frequently, are mounted by more than one module. Thus, it is aimed at satisfying inequality (5.5) with equality and duplicating component types used by frequently occurring mounting tasks first. The current model does not show respect to this situation yet, as there might be many Pareto optima with different left hand sides of inequality (5.5). So how to tackle this? As it is not clear in the first place, whether a component type is assigned to a static or to a dynamic trolley, hard constraints like a lower bound on how often a certain component type must be assigned to different workstations are not very useful here. Clearly, the overall duplication ratio r is bounded by

$$\frac{w_{pm} \cdot \left(|\mathcal{SM}| + |\mathcal{DM}|\right)}{|\mathcal{CT}|} \quad \leq \quad r \quad \leq \frac{w_{pm} \cdot \left(|\mathcal{SM}| + |\mathcal{G}| \cdot |\mathcal{DM}|\right)}{|\mathcal{CT}|}. \tag{5.10}$$

To ensure a high duplication ratio, we may add a penalty term to the objective function. The term

$$s_i^\ell \quad := \quad \frac{p_{i\ell}}{\sum\limits_{j \in \mathcal{SM}} y_{ij} \; + \sum\limits_{j \in \mathcal{DM}_\ell} y_{ij}} \tag{5.11}$$

characterizes the average load of machines, which component type i has been assigned to, for a group ℓ. In order to obtain a a highly balanced distribution, we try to minimize the maximum of s_i^ℓ for all groups and component types. For example that may be done by an additional constraint

$$p_{i\ell} \;\leq\; K \cdot \Big(\sum_{j \in \mathcal{SM}} y_{ij} + \sum_{j \in \mathcal{DM}_\ell} y_{ij} \Big), \text{ for all } i \in \{1, \ldots, m\},\ \ell \in \mathcal{G}. \tag{5.12}$$

Naturally, these constraints are added to a unique setup model as well.

Complexity of Multibin Packing

Multibin Packing is $\mathcal{NP}-$complete. This can be easily seen as follows: consider an instance of 3-PARTITION (GAREY & JOHNSON, 1979) consisting of $3m$ items of length p'_i each, m' bins and a capacity bound B. 3-Partition asks for a distribution of the $3m'$ items to the m' bins such that every bin has exactly load B. Now consider the decision variant of the Multibin Packing Problem from above: let $m := m'$, $n := 3m'$, $p_i := p'_i$, $\ell_{\max} := 1$, $w_{pm} := B$ and $wl_j := B$, for all j. 3-PARTITION is $\mathcal{NP}$-complete in the strong sense. Therefore, it is still $\mathcal{NP}$-complete for values of B that are bounded by a polynomial of m, and the above transformation can be done in polynomial time.

5.2.3 An ILP Model for Partly Combined Processing

In the following, we specify our entire scheduling problem as an integer linear programming (ILP) model for minimizing the makespan of the production of a batch job using partly combined processing. The model is fairly general and may be easily adapted to specific machine types. The general objective in this model is to minimize the overall makespan that consists of processing times of all board types weighted by their production counts and the time needed for changing setups between groups. In this general model, the optimal number of groups will be determined. By choosing parameters for the time needed to change setups sufficiently large, we may also determine the minimum number of groups that still allows a feasible setup.

In the model, we use the terminology as above. $\mathcal{P}$ is the set of PCB types in the batch, $\mathcal{M} = \mathcal{SM} \cup \mathcal{DM}$ the set of (static resp. dynamic) placement modules, $\mathcal{CT}$ the set of component types. In order to model a grouping of PCBs, we establish a set $\mathcal{G}$ of groups. As there are at no time more groups than PCBs, we might set $|\mathcal{G}| := |\mathcal{P}|$ or even less. By a binary assignment variable $x_{p,g}$ it is determined which panel p belongs to which group g. Usually, not every group will be used in a solution. Therefore, by $x_g \in \{0, 1\}$ it is determined, whether a group g is in use or not.

All x are $0/1$–variables, z are nonnegative integral variables, and wl are continuous nonnegative variables. Everything else is input. We will explain the meaning of variables, when they are introduced. For the reader's convenience, a comprehensive glossary of notation used in this section can be found in Appendix A.

Objective function:

$$\text{minimize} \quad \sum_{p \in \mathcal{P}} pc_p \cdot wl_p \; + \; t_{setup}(\sum_{g \in \mathcal{G}} x_g - 1) \tag{5.13}$$

That is, we minimize the sum of workloads wl_p over all board types plus the setup times between groups. pc_p is the number of boards to be produced of type p. By t_{setup} we denote the (constant) time for changing a partial setup, that is, usually the time for unplugging and plugging a trolley. $x_g \in \{0, 1\}$ determines whether a group is in use, i.e. there are PCBs assigned to this group.

General constraints

Each board of type p might be moved several *index steps* $s \in S_p$ forward periodically. The total workload wl_p for a board $p \in \mathcal{P}$ is then composed by the workload $wl_{p,s}$ in each index step $s \in S_p$:

$$wl_p = \sum_{s \in S_p} wl_{p,s}, \quad \forall p \in \mathcal{P}. \tag{5.14}$$

The workload $wl_{p,s}$ for a board p in an index step s is the maximum of workloads over all placement modules plus the time for toolbit exchanges. t_j is the time needed for task j, and $x^{mt}_{j,pm,s} = 1$, if and only if task j is executed on module pm in index step s. The time for changing a toolbit on a module is determined by the parameter t_{ex}. The number of toolbit exchanges is counted by $z^{te}_{s,pm,p}$ (see equation (5.38) at the very end of this model):

$$wl_{p,s} = \sum_{j \in J(p)} t_j \cdot x^{mt}_{j,pm,s} \; + \; t_{ex} \cdot z^{te}_{s,pm,p}, \quad \forall s \in S_p, pm \in \mathcal{M}, p \in \mathcal{P}. \tag{5.15}$$

Each task has to be executed on exactly one module in exactly one index step:

$$\sum_{s \in S_p, pm \in \mathcal{M}} x^{mt}_{j,pm,s} = 1, \quad \forall p \in \mathcal{P}, j \in J(p). \tag{5.16}$$

Setup Constraints

Each task processed on the module pm must be visible for the specific module, and the module has the required component type $ct(j)$ available in its setup. This is stated by $x_{ct(j),pm}$. The boolean predicate $vis(j, pm, s)$ indicates whether task j is visible for module pm in step s. This is part of the input and implicitly given by the index scheme. A task may be executed if and only if both prerequisites, $vis(j, pm, s) = 1$ and $x_{ct(j),pm} = 1$ are fulfilled. For short:

$$x^{mt}_{j,pm,s} \leq vis(j, pm, s) \cdot x_{ct(j),pm}, \quad \forall p \in \mathcal{P}, j \in J(p), s \in S_p, pm \in \mathcal{SM}. \tag{5.17}$$

Analogously, for the tasks on modules with a dynamic setup this has to be satisfied in one group. The $0/1$-variable $x_{p(j),g,ct(j),pm}$ indicates, whether the component type is assigned to a module for a group-board type pair (p, g).

$$x^{mt}_{j,pm,s} \leq vis(j, pm, s) \cdot \sum_{g \in \mathcal{G}} x_{p(j),g,ct(j),pm}, \; \forall p \in \mathcal{P}, j \in J(p), s \in S_p, pm \in \mathcal{DM}. \tag{5.18}$$

A component type must be available in group $g \in \mathcal{G}$ if it is used on at least one board type $p \in \mathcal{P}$ that belongs to this group:

$$x_{p,g,ct,pm} \leq x_{g,ct,pm}, \quad \forall p \in \mathcal{P}, g \in \mathcal{G}, ct \in \mathcal{CT}, pm \in \mathcal{DM}. \tag{5.19}$$

If the dynamic module pm is equipped with component type ct in group g for a board type p which belongs to this group, then $x_{p,g,ct,pm} = 1$. Also, $x_{g,ct,pm} = 1$ if the dynamic module pm is equipped with component type ct in group g.

A component type is only available for a group-board type pair if the board type p belongs to group g, i.e. $x_{p,g} = 1$:

$$x_{p,g,ct,pm} \leq x_{p,g}, \quad \forall p \in \mathcal{P}, g \in \mathcal{G}, ct \in \mathcal{CT}, pm \in \mathcal{DM}. \tag{5.20}$$

Grouping Constraints

For each placement module, the sum of the widths of the associated component types does not exceed its capacity:

$$\sum_{ct \in \mathcal{CT}} w_{ct} \cdot x_{ct,pm} \leq w_{pm}, \quad \forall pm \in \mathcal{SM}, \tag{5.21}$$

where w_{ct} determines the width of component type ct and w_{pm} the width of module pm. The same applies to modules with a dynamic setup in every group:

$$\sum_{ct \in \mathcal{CT}} w_{ct} \cdot x_{g,ct,pm} \leq w_{pm}, \quad \forall g \in \mathcal{G}, pm \in \mathcal{DM}. \tag{5.22}$$

If p belongs to group g, then $x_{p,g} = 1$. Every board type belongs to exactly one group:

$$\sum_{g \in \mathcal{G}} x_{p,g} = 1, \quad \forall p \in \mathcal{P}. \tag{5.23}$$

Group $g \in \mathcal{G}$ must be used if at least one board type $p \in \mathcal{P}$ belongs to it:

$$x_{p,g} \leq x_g, \quad \forall p \in \mathcal{P}, g \in \mathcal{G}. \tag{5.24}$$

Finally, we postulate an ordering of the sequence of groups to eliminate permutations. This is not essential for the model, but narrows the solution space by preventing an exponential number of symmetric solutions.

$$x_{g_1} \geq x_{g_2} \geq \cdots \geq x_{g_k} \tag{5.25}$$

Note that constraints (5.21) and (5.22) are essential for both, the grouping as well as the setup.

Toolbit Compatibility Constraints

The boolean predicate $f^{ct}_{o,ct} \in \{0,1\}$ determines iff a toolbit $o \in O$ is compatible with a component type $ct \in \mathcal{CT}$. The $0/1$-variable $x^t_{o,s,pm,p(j)}$ indicates whether a toolbit o is used in index step s at module pm for the board type $p(j)$. If a mounting task $j \in J$ is executed on pick-and-placement module $pm \in \mathcal{M}$ and index step $s \in S$, then a compatible toolbit type must be used:

$$x^{mt}_{j,pm,s} \leq \sum_{o \in O} f^{ct}_{o,ct(j)} \cdot x^t_{o,s,pm,p(j)}, \quad \forall j \in J, pm \in \mathcal{M}, s \in S. \tag{5.26}$$

At least one toolbit type is associated with every combination of board type p, module pm and index step s:

$$\sum_{o \in O} x^t_{o,s,pm,p} \geq 1, \quad \forall s \in S, pm \in \mathcal{M}, p \in \mathcal{P}. \tag{5.27}$$

Toolbit Exchange Constraints

There is a first toolbit type associated with every combination of board type p, module pm and index step s:

$$\sum_{o \in O} x^{f}_{o,s,pm,p} = 1, \quad \forall s \in S, pm \in \mathcal{M}, p \in \mathcal{P}. \tag{5.28}$$

A toolbit can only be first toolbit if it is assigned at all:

$$x^{f}_{o,s,pm,p} \leq x^{t}_{o,s,pm,p}, \quad \forall o \in O, s \in S, pm \in \mathcal{M}, p \in \mathcal{P}. \tag{5.29}$$

There is a last toolbit type associated with every combination of board type p, module pm and index step s:

$$\sum_{o \in O} x^{l}_{o,s,pm,p} = 1, \quad \forall s \in S, pm \in \mathcal{M}, p \in \mathcal{P}. \tag{5.30}$$

A toolbit can only be last toolbit if it is assigned at all:

$$x^{l}_{o,s,pm,p} \leq x^{t}_{o,s,pm,p}, \quad \forall o \in O, s \in S, pm \in \mathcal{M}, p \in \mathcal{P}. \tag{5.31}$$

The $0/1$-variable $x^{onetbt}_{s,pm,p}$ indicates whether exactly one toolbit type is assigned with the combination of index step s, module pm and board type p. We make sure that $x^{onetbt}_{s,pm,p} = 1$ iff exactly one toolbit type is assigned:

$$n_o(1 - x^{onetbt}_{s,pm,p}) \geq \sum_{o \in O} x^{t}_{o,s,pm,p} - 1, \quad \forall s \in S, pm \in \mathcal{M}, p \in \mathcal{P}, \tag{5.32}$$

$$n_o \cdot x^{onetbt}_{s,pm,p} \geq 2 - \sum_{o \in O} x^{t}_{o,s,pm,p}, \quad \forall s \in S, pm \in \mathcal{M}, p \in \mathcal{P}. \tag{5.33}$$

We set $x^{prec}_{o,s,pm,p} = 1$ if o is the first *or* the last toolbit associated with module pm, index step s and board type p:

$$x^{prec}_{o,s,pm,p} = x^{l}_{o,s-1,pm,p} \vee x^{f}_{o,s,pm,p}, \quad \forall o \in O, pm \in \mathcal{M}, p \in \mathcal{P}, s = 2, \ldots, n_s, \tag{5.34}$$

$$x^{prec}_{o,1,pm,p} = x^{l}_{o,n_s,pm,p} \vee x^{f}_{o,1,pm,p}, \quad \forall o \in O, pm \in \mathcal{M}, p \in \mathcal{P}. \tag{5.35}$$

We set $x^{f \vee l}_{o,s,pm,p} = 1$ iff toolbit o is the first or the last associated toolbit with module pm, index step s and board type p:

$$x^{f \vee l}_{o,s,pm,p} = x^{f}_{o,s,pm,p} \vee x^{l}_{o,s,pm,p}, \quad \forall o \in O, pm \in \mathcal{M}, p \in \mathcal{P}, s \in S. \tag{5.36}$$

We set $x^{f==l}_{s,pm,p} = 1$ iff there are at least two toolbits associated with module pm, index step s and board type p, and the first and the last assigned toolbits are of equal type:

$$x^{f==l}_{s,pm,p} = 2 - \sum_{o \in O} x^{f \vee l}_{o,s,pm,p} - x^{onetbt}_{s,pm,p}, \quad \forall s \in S, pm \in \mathcal{M}, p \in \mathcal{P}. \tag{5.37}$$

The number of toolbit changes is the sum of the associated toolbit types minus 1. This number has to be increased by one if more than one toolbit is assigned and the first and last toolbit are the same. Moreover, it must further increased by one if the first toolbit in this index step is different from the last toolbit in the preceding index step:

$$z^{te}_{s,pm,p} = \sum_{o \in O} x^{t}_{o,s,pm,p} + \sum_{o \in O} x^{prec}_{o,s,pm,p} + x^{f==l}_{s,pm,p} - 2, \quad \forall s \in S, pm \in \mathcal{M}, p \in \mathcal{P}. \tag{5.38}$$

Solvability of the presented model

We have tried to solve our test cases described in Section 5.5 with CPLEX 10.1 and different parameter settings. For computation, we used a model that was slightly simplified: we do not impose the toolbit compatibility and toolbit exchange constraints (5.26)–(5.38). Even with this simplified model, in all test cases, we did not obtain utilizable results after days of computation. For the largest of our test cases, we did not even get bounds after 2 weeks.

5.2.4 The Machine Setup Problem

Let $\mathcal{DM}_\ell$ be the set of placement modules with a dynamic setup during the processing of group G_ℓ with respect to their component assignments.

Definition 5.1 (Partial Setup)**.** *Let G_ℓ be a group of board types that have to be produced on an assembly line $\mathcal{M}$. A* partial setup *of $\mathcal{M}$ with respect to group G_ℓ is an assignment of component types to placement modules $\mathcal{SM} \cup \mathcal{DM}_\ell$ in such way that every component type used by board types in G_ℓ is assigned to at least one module from $\mathcal{SM} \cup \mathcal{DM}_\ell$. In particular, the partial setup is called* feasible *if and only if*

- *for the group G_ℓ and all placement modules $pm \in \mathcal{SM} \cup \mathcal{DM}_\ell$: the assigned component types do not exceed the placement module's feeder capacity $\lambda_{\max}$,*
- *for all PCBs $p \in G_\ell$: component types are assigned to $\mathcal{SM} \cup \mathcal{DM}_\ell$ in such a manner that every task on p using a specific component type is visible on at least one placement module, this module has a specific toolbit that is capable of mounting the component type, and the component type itself is assigned to the specific placement module's setup in the appropriate group G_ℓ.*

Definition 5.2 (Feasible Partly-Combined Setup)**.** *Given a batch job $\mathcal{W}$ and a grouping $\mathcal{G}$ of board types of $\mathcal{W}$ into groups $\{G_1, \ldots, G_c\}$, a* feasible partly-combined setup *is a feasible partial setup for $\mathcal{SM} \cup \mathcal{DM}_\ell$ for every group $G_\ell \in \{G_1, \ldots, G_c\}$.*

5.2.5 The Job Grouping Problem

We are given a batch job $\mathcal{W}$ that consists of a set $\mathcal{P}$ of PCB types $\{p_1, \ldots, p_k\}$, and a machine $\mathcal{M}$. $\mathcal{M}$ consists of static modules $\mathcal{SM}$ holding a static component setup and dynamic modules $\mathcal{DM}$ holding a dynamic component setups. There are some questions with respect to partly-combined processing: (1) How many groups are needed at least, such that a job can be processed at all? In the literature, this question is widely understood as the Job Grouping Problem. In view of partly-combined processing, this question may only be interesting from a theoretical point of view: usually, the makespan that results from a grouping with more groups than the minimal number is considerably better than the one resulting from a minimal grouping. This is due to the fact that the workload can be balanced more efficiently between individual modules as frequently used component types might be duplicated to more than one placement module. This is also a reason why partly-combined processing with exchangeable trolleys is such promising: even if it is possible to process a job with ℓ groups, spending more than ℓ groups may lead to a much better makespan. Time-consuming setup changes are avoided at the same time. Thus, we understand as the Job Grouping Problem the question: (2) What is the best grouping, i.e. the grouping that yields a small makespan on one hand while spending a reasonable number of partial setup changes on the other hand? In the following, we will analyze the following decision version of the Job Grouping Problem.

Definition 5.3 (Job Grouping Problem - decision version)**.** *Given a batch job $\mathcal{W}$ consisting of a set $\mathcal{P}$ of different PCB types, and given a number of groups ℓ, is there a feasible machine setup for a grouping with ℓ groups such that the feeder capacity restrictions on dynamic and static modules are satisfied?*

In Section 5.4.3, we will give a guideline how to choose the number of groups in order to ensure an efficient workload balancing. There might be additional constraints that have to be satisfied in order to allow a feasible setup as mentioned in Section 5.2.4.

5.3 Complexity of Job Grouping and Machine Setup

In this section, we will show that the Job Grouping and Machine Setup problem is strongly $\mathcal{NP}$–complete by a polynomial transformation from a strongly $\mathcal{NP}$–complete problem. By the Job Grouping and Machine Setup problem, we denote the slightly simplified problem "Given any positive integer ℓ, is there a feasible machine setup for a grouping with ℓ groups such that the feeder capacity restrictions are satisfied?". We will distinguish statically and dynamically assigned component types as well.

There are a few reasons that make polynomial transformations from $\mathcal{NP}$-complete problems such as SET PACKING or EXACT COVER to this general formulation of the Job Grouping and Machine Setup problem difficult: on one hand, the subsets occurring in these problems may strongly vary in cardinality, and on the other hand, elements usually occur in multiple sets, but with a varying frequency. So the straightforward approach to express disjointness of sets by a capacity constraint that again is imposed on the number of component types does certainly fail. However, the two above problems might be used if we do not take the whole complexity of the Job Grouping and Machine Setup problem into account, but handle either the component-to-PCB or the PCB-to-cluster affiliations implicitly. The reduction from SET PACKING described in BALAKRISHNAN & VANDERBECK (1999) imposes the capacity constraint on the number of components contained in one *product*. The authors denote by a *product* either a single PCB or a group of several boards. Generally the capacity constraint will not hold for a group of products. Therefore a grouping must be implicitly contained within the set of products. Unfortunately, this entails the fact, that a solution to the corresponding set packing instance with ℓ disjoint sets does not yield a grouping into ℓ groups, but usually more.

We have chosen another option, namely the reduction from EXACT COVER, with the goal to have a one-to-one correspondence from the number of sets to the number of groups. The essential drawback here is that we have to work exclusively with component type sets of groups of PCBs, and it is not clear a priori to which dedicated PCB type they might belong. However, we will present the transformation in the following as it is very intuitive, and not too large.

Finally, in Section 5.3.2, we reduce from strongly $\mathcal{NP}$-complete 3-PARTITION which gives us the additional structure mentioned above, namely the predicate that all of the sets have exactly the same size.

5.3.1 Reduction from EXACT COVER

Definition 5.4 (Exact Cover)**.** *Given a universe of elements $\mathcal{P}$ and a collection $\mathcal{C}$ of sets, an* exact cover *is a subcollection $\mathcal{C}^\star \subseteq \mathcal{C}$ such that every element from $\mathcal{P}$ is contained in exactly one set of $\mathcal{C}^\star$.*

EXACT COVER consists in the problem to decide for given $\mathcal{P}$ and $\mathcal{C}$ whether an exact cover exists. EXACT COVER itself is known to be $\mathcal{NP}$-complete, it might be reduced from 3-SAT (GAREY & JOHNSON, 1979; KARP, 1972).

In the Job Grouping and Machine Setup instance, we will distinguish statically and dynamically assigned component types by a binary variable s_{ct}: for all component types $ct_\ell \in \mathcal{ST}$ (ct_ℓ is assigned

statically), we set $s_{ct_\ell} = 1$, and for all $ct_\ell \in \mathcal{DT}$ (ct_ℓ is assigned dynamically), we set $s_{ct_\ell} = 0$. Furthermore, we state a machine wide capacity bound C instead of stating the constraint for every placement module as it is done in (5.21) in the ILP model.

More formally, given a set $\mathcal{P}$ of PCBs, a set of component types $\mathcal{CT}$, subsets $\mathcal{CT}(i) \subseteq \mathcal{CT}$ for each group $i \in \{1, \ldots, |\mathcal{C}|\}$, C feeder slots in total, and a natural number ℓ. We are looking for a grouping $\mathcal{C}^\star \subseteq \mathcal{C}$ with $|\mathcal{C}^\star| = \ell$ such that

$$\sum_{ct \notin \mathcal{CT}(i)} s_{ct} + \mathcal{CT}(i) \leq C \text{ for all } i \,:\, C_i \in \mathcal{C}^\star, \tag{5.39}$$

and $C_k \cap C_\ell = \emptyset$ for all $C_k, C_\ell \in \mathcal{C}^\star$, $k \neq \ell$.

Given any instance $(\mathcal{P}, \mathcal{C})$ of EXACT COVER, consider the corresponding Job Grouping and Machine Setup instance with a set $\mathcal{P}$ of PCBs, a set $\mathcal{C}$ of groups, a set $\mathcal{CT} := \{1, \ldots, 2 \cdot |\mathcal{C}|\}$ of component types used for all PCBs. Furthermore, there are component sets $\mathcal{CT}(i)$ for each group i defined by

$$\mathcal{CT}(i) := \{j \,:\, C_i \cap C_j = \emptyset\} \cup \{j + |\mathcal{C}| \,:\, C_i \cap C_j \neq \emptyset\} \text{ for all } i \in 1, \ldots, |\mathcal{C}|. \tag{5.40}$$

Thus, every group i contains exactly $|\mathcal{C}|$ component types. Moreover, we set the total number of feeder slots to $C := |\mathcal{C}| + 1$.

Let $\boldsymbol{x} \in \{0,1\}^{|\mathcal{C}|}$ be a solution to the EXACT COVER instance $(\mathcal{P}, \mathcal{C})$. We choose the set of statically assigned component types $\boldsymbol{s} := \boldsymbol{x}$, that is exactly the indices of sets chosen in a solution form the set of statically assigned component type indices. By construction, every group contains already $C - 1$ component types. As only indices of groups that are disjoint to a group i chosen in the solution are at the same time component indices in $\mathcal{CT}(i)$, and all indices of chosen groups except i are part of each $\mathcal{CT}(i)$, we just have to add i itself to the sum of statically assigned component types in (5.39), and the capacity bound C is met. Any infeasible solution to $(\mathcal{P}, \mathcal{C})$ would yield an exceedance of C as more than one statically assigned components would have to be added to the sum in (5.39). These exceedances are exactly occurring in those groups that are responsible for $\mathcal{C}^\star$ being not disjoint. The other way round, a solution to the Job Grouping and Machine Setup problem violating (5.39) cannot yield a feasible solution to EXACT COVER, because the chosen subsets are not disjoint.

As mentioned earlier, the problem with the transformation presented above is the fact that the component type sets $\mathcal{CT}(i)$ are chosen in affiliation with a specific group, and might not be transferred into PCB specific sets. For this reason, we will present another transformation from 3-PARTITION.

5.3.2 Reduction from 3-PARTITION

Definition 5.5 (3-PARTITION)**.** *Given a multiset $\mathcal{P}$ of integers. Is there a partition of $\mathcal{P}$ into triples such that all triples have the same sum?*

3-PARTITION is strongly $\mathcal{NP}$-complete (GAREY & JOHNSON, 1979). It still remains $\mathcal{NP}$-complete even if the integers in S are bounded by a polynomial in $|\mathcal{P}|$.

Given an instance $\mathcal{P}$ of 3-PARTITION consisting of $3m$ integers $\{p_1, \ldots, p_{3m}\}$. Consider the corresponding Job Grouping and Machine Setup instance with a set of $3m$ PCBs $\{1, \ldots, 3m\}$, a set $\mathcal{CT}$ of $D + 1$ component types

$$\{ct_1, \ldots, ct_{D+1}\}, \text{ where } D := \sum_{i=1}^{3m} p_i.$$

Further given sets $\mathcal{CT}(\ell) \subseteq \mathcal{CT}$ of component types used by PCB ℓ by

$$\mathcal{CT}(\ell) := \{ct_{1+I_{\ell-1}}, \ldots, ct_{I_\ell}\} \cup \{ct_{D+1}\}, \text{ for all } \ell \in \{1, \ldots, 3m\}, \tag{5.41}$$

where I_ℓ is the sum of the first ℓ items $p_1, \ldots, p_\ell$ from $\mathcal{P}$, and $I_0 := 0$. The capacity bound is given by $C := D/3 + 1$. Moreover, let $\mathcal{DT} := \{ct_1, \ldots, ct_D\}$ be the set of dynamically assigned component types, and $\mathcal{ST} := \{ct_{D+1}\}$ the set of statically assigned ones.

By construction, the ℓ-th PCB contains exactly $p_\ell + 1$ component types, from which only one is statically assigned. Any feasible 3-partition of $\mathcal{P}$ transforms immediately to a feasible solution of the constructed Job Grouping and Machine Setup instance with m groups: every group contains exactly $C - 1$ dynamically assigned and one statically assigned component type. This exactly meets the capacity bound C.

The other way round, any solution to the constructed Job Grouping and Machine Setup instance with m groups yields a feasible solution to 3-PARTITION as the capacity bound C must be exactly met in a grouping with m groups. Otherwise there would be a group with less than C components. Clearly, as all component sets of PCBs are pairwise disjoint except for component ct_{D+1}, there would be at least one group violating the bound C. Since the total number of dynamically assigned component types in each group is D, the sum of the each corresponding set from $\mathcal{P}$ is D as well.

From the above considerations, we can make the concluding statement:

Theorem 5.6. *The Job Grouping and Machine Setup problem is $\mathcal{NP}$–complete in the strong sense.*

5.4 A novel approach to the Job Grouping and Setup Computation Problem for Partly-Combined Processing

In this section, we present a novel approach to jointly solving the job grouping and setup problem for partly combined processing. The main goal of our approach is to allow an optimal workload balancing in order to minimize the makespan. For this reason, frequently used component types have to be installed on more than one placement module. We will call this a *duplication* of component types. In order to achieve this, feeder capacity constraints are incorporated during the early grouping phase yet. This also allows to detect infeasibilities at the early grouping stage. Moreover, component assignments are balanced between the static and the dynamic part of the setup during grouping. Our approach divides into three parts: first, the determination of a grouping allowing the largest possible degree of freedom to the subsequent processing; second, the computation of an initial feasible partly-combined setup; and third, the expansion of the setup by duplication of component types in order to balance the workload.

5.4.1 Merge and Distribute

There are two fundamental decisions to make during the grouping process: (1) which board types are combined and (2) which component types are placed on the static and which on the dynamic part of the setup. This is mainly influenced by the capacity restrictions of the placement modules. In a sequential process to build up groups from singletons by merging them together, it is not clear, whether such a grouping will ever yield a feasible setup, and if so, how the grouping itself affects the setup in terms of a possible load balancing which again has effects on the makespan. In order to reduce the computation times for the whole optimization process, we do not intend to compute a complete setup and a mounting task distribution each time we have computed a grouping of board

types in order to determine feasibility. Therefore, we are tempted to bias these consequences indirectly by maximizing the probability that a feasible setup exists during the grouping yet. This is achieved by integrating the partitioning into static and dynamic component types into the grouping process: Let $\mathcal{ST}$ be the set of component types that is assigned to modules with a static feeder setup, and $\mathcal{DT}_\ell$ be the set of component types that is assigned to modules with an exchangeable (dynamic) setup for the group G_ℓ. Every module has a fixed maximum capacity of at most $\lambda_{\max}$ different component types. This is due to the construction of the machine. Our idea now is to assume – at least for the grouping phase – that every module may virtually hold less component types, namely at most $\lambda < \lambda_{\max}$. For this reason, on every module, some slots remain unused in order to duplicate component types later. An initial λ is determined as follows: there are $|\mathcal{CT}|$ different component types in a batch, that have to be distributed over $|\mathcal{SM} \cup \mathcal{DM}|$ modules in total, of which $|\mathcal{DM}|$ of them feature a dynamic setup. Assuming that the dynamically placed component types in one group are unique for the whole setup, i.e. are occurring in only one $\mathcal{DT}_\ell, \ell \in \{1, \ldots, g\}$, there have to be

$$\tilde{\lambda}_g := \frac{|\mathcal{CT}|}{|\mathcal{SM}| + g \cdot |\mathcal{DM}|} \tag{5.42}$$

component types placed on each module on average when using g groups. This gives us both, a lower bound on the mean assignment load, as well as an upper bound $\tau_u := \lambda_{\max}/\tilde{\lambda}_g$ on the mean duplication ratio that is obtainable with g groups. Thus, we approximately know how many groups we have to spend in order to obtain a duplication rate we aim at. As the sets of component types assigned dynamically are not disjoint, we adjust λ during the algorithm. We start with $\lambda := \lceil \tilde{\lambda}_g \rceil$ and increase λ whenever necessary.

Algorithm 9 `bool distributeComponentTypes`

Input: Grouping $\mathcal{G}$, Group G_ℓ, set $\mathcal{ST}$, sets $\mathcal{DT}_j$, parameter λ
Modifies:set $\mathcal{ST}$ of static components , sets $\mathcal{DT}_j$ of dynamic components

forall component types ct_i in $\mathcal{DT}_\ell$ of group G_ℓ **do**
  let $occ(ct_i)$ be the number of PCBs in $\bigcup G_j,\ G_j \in \mathcal{G}$, containing ct_i
sort component types ct_i decreasingly by $occ(ct_i)$
while $|\mathcal{DT}_\ell| > \lambda \cdot |\mathcal{DM}|$ **do**
  choose component ct_ℓ with highest occurrence $occ(ct_\ell) \geq occ(ct_i)$
  forall groups $G_j \in \mathcal{G}$ containing ct_ℓ
    move component type ct_ℓ from $\mathcal{DT}_j$ to $\mathcal{ST}$
  if $|\mathcal{ST}| > \lambda \cdot |\mathcal{SM}|$ **then return** false
return true

At the beginning of Algorithm 10, each board type i constitutes its own group G_i, and all of its component types are virtually assigned to its dynamic component set $\mathcal{DT}_i$. Usually, this start configuration is infeasible. For each board type, the algorithm determines how many component types must be moved to the static part $\mathcal{ST}$ of the setup in order to make the configuration feasible subject to the virtual capacity restrictions λ of the placement modules (cf. equation (5.42)). Each component type in a specific group is regarded and its occurrence in all groups is determined. Then, exactly those component types recurring in most of the other groups are moved to the static part of the setup as long as the capacity bounds on the dynamic part are violated. We will call this method `distributeComponentTypes` (cf. Algorithm 9).

In each iteration of the grouping algorithm, two groups are merged. For this reason, a similarity measure is needed to determine the two groups to merge. From the variety of similarity measures

presented by STEINBACH ET AL. (2000), a variant of *Jaccard's coefficient* empirically turned out to be most appropriate for our application. First, we will give the original version of Jaccard's coefficient, and then the modified one.

Definition 5.7 (Jaccard's coefficient)**.** *Given two groups A and B with their sets of component types $\mathcal{CT}_A$ and $\mathcal{CT}_B$,* Jaccard's coefficient *is defined by*

$$\Delta(A,B) := \frac{|\mathcal{CT}_A \cap \mathcal{CT}_B|}{|\mathcal{CT}_A \cup \mathcal{CT}_B|}.$$

Algorithm 10 `Grouping MergeAndDistribute`

Input: Grouping $\mathcal{G}$ with sets $\mathcal{ST}$ and $\mathcal{DT}_j$ for each group $j \in \mathcal{G}$, $\lambda, \lambda_{\max}, d$
Output: Grouping $\mathcal{G}$ with set $\mathcal{ST}$,and sets $\mathcal{DT}_j$ for each group $j \in \mathcal{G}$

```
calculate similarities Δ_d(A,B) for all group pairs (A,B) ∈ G × G, A ≠ B
while |G| > d (desired number of groups) do
    for all pairs (A,B) ∈ G × G, A ≠ B sorted decreasingly by Δ_d(A,B)
        if (A,B) is marked unmergeable then continue
        C ← merge(A,B)
        if distributeComponentTypes(C,λ) then
            G ← G ∪ C
            G ← G \ {A,B}
            for all groups X ∈ G \ C calculate Δ_d(X,C)
            break
        else
            mark (A,B) unmergeable
    if all pairs (A,B) ∈ G × G, A ≠ B are marked unmergeable then
        λ ← λ + 1
        if λ > λ_max then return G // no feasible solution found
        mark all pairs (A,B) ∈ G × G, A ≠ B mergeable
        MergeAndDistribute(G, λ,d)
return G
```

In our approach, we distinguish statically assigned from dynamically assigned component types. Moreover, component types are moved only from the dynamically assigned part to the statically assigned part of the setup, and not vice versa. So the number of component types on the dynamically assigned part decreases. Using Jaccard's coefficient, two groups have always the same similarity value, independently from what has already moved to the statically assigned part. This effect is not desirable. So, the idea is to take only dynamically assigned component types, $\mathcal{DT}_A$ and $\mathcal{DT}_B$, into account in order to not dilute the similarity measure. Therefore, we define the similarity coefficient as:

Definition 5.8 (Dynamic Jaccard's coefficient)**.** *Given two groups A and B with their dynamic sets of component types $\mathcal{DT}_A$ and $\mathcal{DT}_B$, the* dynamic Jaccard's coefficient *is defined by*

$$\Delta_d(A,B) := \frac{|\mathcal{DT}_A \cap \mathcal{DT}_B|}{|\mathcal{DT}_A \cup \mathcal{DT}_B|}.$$

For merging, we choose the two groups A and B that provide the highest similarity value according to $\Delta_d(A,B)$. After ensuring that there is a successful distribution of component types according to

Algorithm 9, the groups as well as their sets of dynamic component types are merged, and Algorithm 9 is called for this merged set. If there is no feasible distribution of components, the second most similar groups are tried to merge, and so on. During this process, it may happen that there is no merge operation of any of the groups in $\mathcal{G}$ that allows a feasible distribution of components. In this case, λ_g must be increased in order to allow more merge operations. The procedure is repeated until either a chosen number of groups is obtained, or λ has reached $\lambda_{\max}$. A third alternative for a termination criteria is to give a bound $\lambda_d < \lambda_{\max}$ in order to ensure a specific duplication ratio not lower than a given bound. With a given λ_d the optimal number of groups will be determined implicitly (cf. equation (5.42)).

If the grouping algorithm has not attained the given number of groups yet, and $\lambda = \lambda_{\max}$ has been reached, the procedure did not find a feasible grouping with the desired number of clusters.

5.4.2 Setup Computation for Partly-Combined Processing

There are several approaches that compute feasible setups for single boards or groups first, and then merge them in order to obtain a feasible setup for the complete batch (cf. AMMONS ET AL. (1997)). There are usually many repair operations necessary to gain feasibility, and it is not clear if a feasible setup does exist at all. Especially in our scenario which comprises a lot of board types, this approach is not practicable as it cannot be guaranteed that feasible setups exit at all. Depending on the machine used, gaining feasibility of setups is highly complex: There are multiple restrictions, most importantly visibility and toolbit compatibility constraints. Some of them might not even be linear, or can only be modeled as such with a maximum of effort (cf. our ILP model in Appendix 5.2.3). Therefore, we will confine ourselves to computing feasible setups solely. We first compute a feasible initial setup. Then, in a second step, the setup is extended for workload balancing, always preserving feasibility. This two stage approach benefits feasible setups even in case the batch jobs are very tight in terms of the mere number of distinct component types.

Besides the grouping itself, the grouping algorithm delivers the sets $\mathcal{ST}$ and $\mathcal{DT}_\ell$, which we take as a suggestion whether to place component types onto static or dynamic modules. This is not a hard constraint, but normally a satisfiable one: Recall that we have assumed the number of component types to be assigned to each module λ is lower than its intrinsic capacity restriction $\lambda_{\max}$. There might be constraints that do not allow the preferred assignment. Thus, our first attempt is to assign a component type to that kind of module that was identified by the grouping algorithm, either static or dynamic. If this fails, the component type is assigned to the other type. Note that during the duplication phase, component types might be duplicated to the other type of trolley as well in order to achieve better load balancing.

We use Algorithm 11 to assign component types to modules. As mentioned above, we only make feasible assignments with respect to Definitions 5.1 and 5.2. We use a *best-fit* (BF) strategy (HOROWITZ & SAHNI, 1978; LEMAIRE ET AL., 2005) from BIN PACKING, which always assigns a component type to the module with lowest workload, always conserving feasibility. Regardless of this, the solution might depend on the sequence of component types assigned and thus of the processing sequence of groups, and the sequence of PCBs in each group, respectively. To weaken this effect, we sort the board types in decreasing order by their production counts multiplied by their total number of mounting tasks, and process them in this order. This is called *best-fit decreasing* strategy (BFD). For the general BIN PACKING PROBLEM, there is a worst-case estimation on the quality of results for the best fit decreasing strategy that is not too bad and there is even a PTAS based on BFD (LEMAIRE ET AL., 2005). Basically, BFD with board types sorted decreasingly by their workload is done in Algorithm 12. In some applications, the global occurrence of the used component types in a batch job may be the sorting criterion of choice: Algorithm 13 assigns component types sorted in decreasing order by their global frequency

Algorithm 11 `Assign`(ct, p, pc)

Input: component type ct, board type p, workload pc of ct on p
Requires: set of static components $\mathcal{ST}$, sets of dynamic components $\mathcal{DT}_j$
Output: assignment of ct to a module $m \in \mathcal{SM} \cup \mathcal{DM}_{G_\ell}, p \in G_\ell$

if component type ct is already assigned to a module $m \in \mathcal{SM} \cup \mathcal{DM}_{G_j},\ \forall G_j \in \mathcal{G}$ **then**
  assign workload pc of ct on board type p to module m
  break
according to assignment preference of ct from the grouping set either $\mathcal{PPM} \leftarrow \mathcal{SM}$ or $\mathcal{PPM} \leftarrow \mathcal{DM}_{G_\ell}$, where G_ℓ is the group, ct is part of
sort $\mathcal{PPM}$ ascendingly by assigned workload
if getFirst$(\mathcal{PPM})$ is empty **then** assign a compatible toolbit
ASSIGNED $\leftarrow$ false
forall $m \in \mathcal{PPM}$ **do**
  if m is full **then continue**
  if all tasks with component ct on board p are visible on m **do**
    if m contains a compatible toolbit **then**
      assign ct to module m
      add pc to workload of module m
      ASSIGNED $\leftarrow$ true
      break
if !ASSIGNED
  forall all $m \in \mathcal{PPM}$
    if m is full **then continue**
    if all tasks using ct on board type p are visible on m **then**
      assign a compatible toolbit to m
      assign ct to module m
      add pc to workload of module m
      break

Algorithm 12 `BoardwiseInitialSetup`

let $\mathcal{P}$ be the set of board types, sorted in decreasing order by their workload (mounting tasks times production count)

forall all $p \in \mathcal{P}$ (in this order !) **do**
  forall ct used on p sorted decreasingly by frequency of occurrence $occ(pt)$
    `Assign`(ct, p, pc)

Algorithm 13 `ComponentType-WiseInitialSetup`

let $\mathcal{PT}$ be the set of component types, sorted in decreasing order by their global frequency of occurrence

forall all $pt \in \mathcal{PT}$ (in this order !) **do**
  forall board types p, on which pt is mounted $occ(pt) > 0$ times
    `Assign`(ct, p, pc)

of occurrence in the batch job.

5.4.3 Duplication and Workload Balancing

The quality of a setup depends mainly on how the workload can be balanced by the subsequent distribution of tasks. Workload balancing can only be achieved to a satisfactory extent if frequently used component types are assigned to multiple modules exhaustively. This is what we call *duplication* of component types. Duplication is essential to obtain a good makespan of a batch. Our main algorithmic idea is to balance the workload per board type as this is the crucial criterion for the makespan. To our experience, this gives rise to a higher duplication ratio as well. From equation (5.42) we know that the maximum average duplication ratio τ_{avg} is bounded by

$$\tau_\ell := \frac{\lambda_{\max}}{\lambda_1} \leq \tau_{avg} \leq \frac{\lambda_{\max}}{\lambda_g} =: \tau_u, \tag{5.43}$$

when using g groups. For the duplication algorithm, we need a criterion, which component types are to be duplicated and which not. For this reason, we define a bound per board type by

$$LB(p) := \left\lceil \frac{|J(p)|}{|\mathcal{SM} \cup \mathcal{DM}_G|} \right\rceil, \tag{5.44}$$

where $J(p)$ are the mounting tasks occurring on board type p and G the group to which p belongs. $LB(p)$ denotes the average workload of each module while producing board type p (rounded up to the nearest integer).

Our duplication algorithm (Algorithm 14) passes the board types iteratively and duplicates exactly these component which occur with a higher frequency than than $LB(p)$ and the parameter $minPC$. First, the algorithm duplicates component types that are used by mounting tasks occurring with a frequency higher than $LB(p)$. These are duplicated exclusively to modules with a currently assigned workload lower than $LB(p)$. Usually, we choose the module with the lowest current workload. This is crucial to eliminate sporadic workload peaks on modules which lead to an idling of most other modules, and therefore results in a bad makespan. See Figure 5.3 for an illustration how Algorithm 14 works.

Algorithm 14 `DuplicateSetupBalanced`

forall PCBs p **do**
  $\mathcal{DUP} \leftarrow$ component types ct used by $t(ct)$ tasks with $t(ct) > LB(p)$
    and $t(ct) > minPC$
  forall component types $ct \in \mathcal{DUP}$ **do**
    forall modules $m \in \mathcal{SM}$ sorted increasingly by their workload **do**
      if $load(m) > LB(p)$ **then continue**
      if number of component types on m equals $\lambda_{\max}$ **then continue**
       assign ct to m
       $t(ct) \leftarrow t(ct)/2$
       break

Generally, it is a good strategy to duplicate the component type with the highest production count to the module with the lowest number of assigned components containing a compatible toolbit. In a second phase, toolbit exchanges may be necessary to duplicate component types efficiently. We add the potentially needed toolbits to the specific modules in order to take care that the limit of toolbit

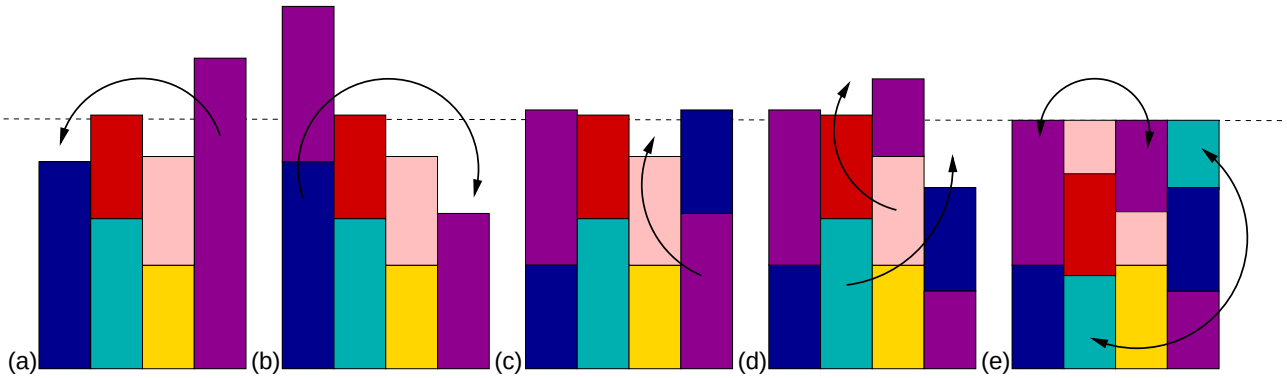


Figure 5.3: Duplication of component types in order to allow efficient load balancing. Each column shows the current workload of one of 4 modules with respect to the assigned component types each of which are colored differently. The dotted line shows the average workload per module $LB(p)$. In subfigures (a) to (e) each, a component type with the highest production count on the module with highest workload is distributed to the module with lowest workload. The workload of the duplicated component is bisected in each case. In this example, we assume a capacity restriction of at most 3 component types per module, so the duplication algorithm terminates at (e). As indicated in (e), the distribution of mounting tasks would balance the workload equally by shifting some tasks from modules with workload exceeding $LB(p)$ to those falling below it. The makespan might actually take the optimal value $LB(p)$.

types a module can really hold is not exceeded. The real toolbit exchanges are then added by the algorithm that distributes the tasks over the modules. This is due to the fact that it is not clear a priori, if a duplication of a component type would yield a lower makespan at all, i.e. if there is a positive tradeoff in terms of makespan when spending the time for an exchange of a toolbit. If there is some feeder space left on the modules after the duplication algorithm, the most frequently used component types in relation to their current duplication ratio are duplicated exhaustively until all feeder slots are occupied.

5.5 Computational Experiments

In order to perform computational experiments, we use a platform that emulates the AX series machines from Philips Assembléon. We implemented their largest machine at this time featuring 20 placement modules in a typical configuration: each 4 modules are supplied by feeders located on one trolley. We allow one or two trolleys, respectively, to change. So in total, 4 or 8 modules will have a dynamic setup in our scenario. Figure 5.1 shows such a machine configuration that uses one exchangeable trolley and Table 5.5 shows the characteristics of our test jobs (number of board types, total number of mounting tasks, required number of different component types, number of different toolbits, and available modules).

We have run several experiments using large scale problem instances from automotive and cellphone manufacturers. To evaluate our results, we implemented an Agglomerative Clustering algorithm for grouping, and the list processing based heuristic from AMMONS ET AL. (1997) for the setup computation. In order to obtain cycle times, the task distribution is done by a max flow algorithm and a subsequent local search (MÜLLER-HANNEMANN ET AL., 2006) in both cases. So differences in cycle times arise exclusively from the board grouping and setup computation.

Unfortunately, our original intention – namely to obtain an exact solution or at least good lower bounds via the ILP model presented in Appendix 5.2.3 – failed due to the mere size of the model.

	Job Characteristics					
Job	Boards	Mounts	Comp's	Tools	Modules	Slots/Comp's
1	42	25840	166	3	20	0.84
2	39	23480	151	3	20	0.92
3	30	20912	155	3	20	0.90
4	23	5781	182	5	20	0.75

Table 5.1: Characteristics of the test jobs originating from automotive and cellphone manufacturers. The last column states the quotient of overall average feeder capacity by the sum of component types using one unique machine setup. Clearly, if this quotient is below 1, there cannot be any feasible setup without setup changes during processing.

Even after days of computation with CPLEX 10.1, the gap was far too large to draw any conclusions from that. In our largest instance we did not even succeed to compute bounds after 2 weeks.

5.5.1 Computational Results

For each of the above jobs, we computed results with a varying number of groups. For each number of groups, we present several cycle time results: On the one hand, we give cycle times for the case of a combined setup, which means that the whole machine setup changes between the processing of two groups. On the other hand, cycle times are given for the partly combined case (PC) with 4 and 8 exchangeable trolleys, respectively. For each of the above cases, we give one cycle time arising from a grouping and setup computed by the Agglomerative Clustering approach, and one cycle time arising from a grouping and setup computed by our novel Merge&Distribute approach.

The cycle times are computed in a normalized way, that is, the batch cycle time consists of the sum of board cycle times under the assumption that the production count for each board type is 1. This is due to the fact that we were not given real production volumes by manufacturers.

Tables 5.2 to 5.5 summarize the results for our four test jobs. The cycle times arising from groupings and setups obtained by our technique are usually considerably smaller than with the Agglomerative Clustering approach. Especially in the interesting cases, namely if the number of groups is quite small, we obtain the largest savings, namely up to 32% of cycle time (cf. Job 1, 7 groups). Another interesting observation is the fact that an average duplication rate of approximately 1.5 to 2.0 seems to be a good choice as the parallelism of the machine is efficiently exploited then. If the duplication rate is smaller than that, the batch job might be feasible, but cycle times increase dramatically when decreasing the number of groups. On the other hand, at a higher duplication rate, the speedup levels off when increasing the number of groups. Consider for example the first job and the configuration with one dynamic trolley: the first feasible solution is obtained with 5 groups, the average duplication is 1.51. Spending one more group saves about 17% of the cycle time, the duplication rate is 1.68. With one group more, we save another 13%, the duplication is at 1.85. Increasing the number of groups, the cycle time slightly increases, the duplication rate is 2.02. When spending more clusters, the cycle time levels off. So interesting configurations range from 5 groups to 7 as they yield a good tradeoff between trolley exchanges and cycle time. In our other test cases, this effect occurs quite analogously. As the computing times are quite small, we recommend a few computations with all reasonable number of clusters, namely those yielding a duplication rate between 1.5 and 2.0, and then choosing a convenient configuration.

	Combined		PC (4 dyn.PMs)		PC (8 dyn.PMs)	
$\|\mathcal{G}\|$	Agglom	M&D	Agglom	M&D	Agglom	M&D
1	-	-	-	-	-	-
2	1917.06	1824.64	-	-	-	-
3	1797.60	1655.55	-	-	2339.57	2297.90
4	1458.59	1345.25	-	-	2120.88	1693.95
5	1456.81	1213.77	-	2411.65	2046.17	1862.03
6	1267.79	1157.70	-	2004.72	1968.59	1486.42
7	1265.93	1213.58	2552.65	1744.56	1897.75	1488.82
8	1275.51	1183.94	2130.76	1767.20	1515.52	1335.81
9	1260.02	1217.36	1883.92	1663.66	1482.12	1422.27
10	1262.38	1207.54	1902.69	1609.62	1474.21	1260.16
11	1264.75	1238.23	1718.04	1604.97	1282.60	1275.30
12	1275.32	1261.98	1604.66	1601.07	1350.69	1248.76
13	1279.25	1261.89	1634.90	1602.49	1372.17	1294.44

Table 5.2: Computed cycle times in seconds for job 1 with 42 board types. Comparative results for combined and partly-combined (PC) setup computations using the Agglomerative Clustering and our Merge&Distribute approach. There are 4 (middle columns) or 8 (right columns) dynamic placement modules (PMs) used, this amounts to one or two dynamic trolleys, respectively.

	Combined		PC (4 dyn.PMs)		PC (8 dyn.PMs)	
$\|\mathcal{G}\|$	Agglom	M&D	Agglom	M&D	Agglom	M&D
1	-	-	-	-	-	-
2	1513.60	1452.93	-	-	2049.28	1945.66
3	1139.47	1061.73	-	-	1868.50	1753.40
4	1138.00	1061.97	-	1945.11	1564.05	1182.09
5	1063.30	1055.79	2074.23	1721.20	1530.46	1216.88
6	1114.72	1025.57	1880.00	1549.74	1261.68	1216.59
7	1098.65	1062.81	1675.36	1418.78	1134.09	1157.92
8	1098.62	1060.19	1644.97	1389.37	1146.09	1154.53
9	1109.09	1084.61	1343.75	1341.83	1136.42	1165.76
10	1115.53	1082.77	1328.12	1307.48	1129.68	1147.66

Table 5.3: Computed cycle times in seconds for job 2 with 39 board types. Comparative results for combined and partly-combined (PC) setup computations using the Agglomerative Clustering and our Merge&Distribute approach. There are 4 (middle columns) or 8 (right columns) dynamic placement modules (PMs) used, this amounts to one or two dynamic trolleys, respectively.

5.5.2 CPU Times

Our code was written in JAVA, and the test sets were run on an Intel Pentium IV with 3.2GHz. We measure CPU times for the whole framework, from the input of data until the output of a cycle time. Depending on the number of clusters, for our largest batch job with 41 board types, CPU times

	Combined		PC (4 dyn.PMs)		PC (8 dyn.PMs)	
$\|\mathcal{G}\|$	Agglom	M&D	Agglom	M&D	Agglom	M&D
1	-	-	-	-	-	-
2	1241.00	1238.33	-	-	1883.47	1656.44
3	1170.42	1014.38	-	-	1521.25	1502.33
4	1027.17	954.66	-	1571.81	1644.03	1291.25
5	1005.73	993.61	1933.49	1454.71	1438.91	1079.02
6	1030.31	943.97	1630.27	1417.27	1154.97	1042.37
7	1002.60	955.59	1383.24	1346.50	1028.99	1037.38
8	988.08	988.07	1568.20	1342.18	1082.51	1005.32
9	994.86	989.99	1303.38	1262.96	1033.49	991.30
10	989.79	980.63	1295.99	1168.76	1039.48	986.22

Table 5.4: Computed cycle times in seconds for job 3 with 30 board types.Comparative results for combined and partly-combined (PC) setup computations using the Agglomerative Clustering and our Merge&Distribute approach. There are 4 (middle columns) or 8 (right columns) dynamic placement modules (PMs) used, this amounts to one or two dynamic trolleys, respectively.

	Combined		PC (4 dyn.PMs)		PC (8 dyn.PMs)	
$\|\mathcal{G}\|$	Agglom	M&D	Agglom	M&D	Agglom	M&D
1	-	-	-	-	-	-
2	302.94	298.43	-	-	-	-
3	293.10	288.36	-	-	428.94	337.81
4	283.40	275.88	-	532.46	425.45	346.23
5	280.45	273.10	-	434.18	386.63	356.96
6	295.17	267.22	-	433.13	396.47	342.80
7	273.27	265.24	427.09	382.58	326.63	293.00
8	268.45	261.45	399.42	391.29	314.22	300.46
9	270.63	259.99	394.71	391.25	325.25	309.22
10	264.04	255.24	387.50	349.69	319.15	314.10

Table 5.5: Computed cycle times in seconds for job 4 with 23 board types. Comparative results for combined and partly-combined (PC) setup computations using the Agglomerative Clustering and our Merge&Distribute approach. There are 4 (middle columns) or 8 (right columns) dynamic placement modules (PMs) used, this amounts to one or two dynamic trolleys, respectively.

vary from 5 minutes to 11 minutes using the Merge&Distribute technique. The CPU times for the Agglomerative Clustering based approach range from 51 to 79 minutes. In most of the cases, we only use between 10 and 20% of the CPU time of the Agglomerative Clustering based approach. In all of our infeasible test cases, infeasibility is detected at an early stage with our technique: Within circa one minute, we find an instance to be infeasible. This is only 1.3 to 1.8% of the time the other approach requires. The savings in CPU time are mainly due to the fact that the groupings generated by our approach are usually feasible. Avoiding unnecessary feasibility checks saves 80 to 90% of the CPU time of the Agglomerative Clustering based approach in the case of a feasible configuration. This

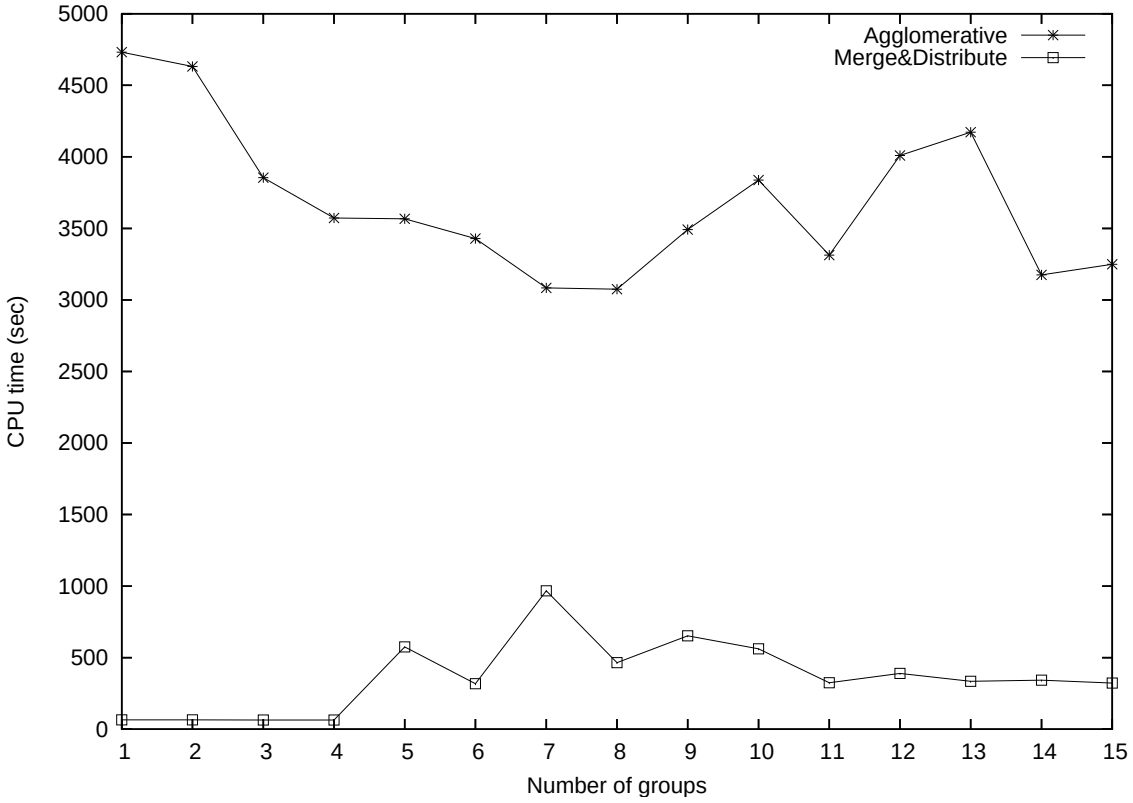


Figure 5.4: CPU times in seconds for the largest test job (41 board types). Merge&Distribute detects infeasibility of groupings very fast. In this case, configurations using 1 to 4 groups yield infeasible setups (cf. Table 5.2).

speedup can be applied to compute several configurations of parameter sets, e.g. varying the number of clusters, the number of exchangeable trolleys, in order to have a better basis for decision-making.

5.6 Summary

In this work, we have presented a novel technique for solving the Job Grouping and Setup Computation problems for partly combined processing in automated PCB manufacturing. Our technique incorporates machine specific constraints already in the early grouping phase. This allows us to compute feasible groupings without the need of checking feasibility by determining a setup each time two groups have been merged. The other way round, infeasibility of the batch job with a given number of groups is detected at an early stage in the framework. At the same time, the prerequisites for the subsequent balancing of workload between the individual modules are accomplished. The developed algorithms for setup computation create feasible setups in a first step. These setups are extended in a second stage in order to allow exhaustive workload balancing. All presented algorithms are very general and thus might be adapted to a wide variety of machine types requiring different side constraints.

As computational experiments using large problem instances from electronics manufacturing demonstrate, the quality of setups is up to 32% better than a standard technique using an Agglomerative Clustering algorithm. Furthermore, feasible configurations with a smaller number of groups could be found. Additionally, our technique is much faster as feasibility is already provided by the job group-

ing algorithm. Thereby, it is possible to compute multiple configurations in the same time in order to support the process of decision-making.

A Glossary of Notation used for the ILP model in Section 5.2.3

For more lucidity, here we give an overview about the complete notation used in the ILP model in Section 5.2.3.

Sets and Constants

- $\mathcal{P} = \{p_1, p_2, \ldots, p_k\}$ denotes the set of given panels
- $\mathcal{G} = \{g_1, g_2, \ldots, g_k\}$ denotes the set of (possible) groups
- $\mathcal{CT}$ denotes the set of component types
- $S = \{1, 2, \ldots, n_s\}$ denotes the set of index steps
- $\mathcal{M}$ denotes the set of pick-and-placement modules
- $\mathcal{DM}$ denotes the subset of pick-and-placement modules with a dynamic machine setup
- $\mathcal{SM}$ denotes the subset of pick-and-placement modules with a static machine setup
- O denotes the set of available toolbit types
- $n_o = |O|$
- J denotes the set of all mounting tasks (over all panels)
- $J(p)$ denotes the set of mounting tasks (jobs) of panel $p \in \mathcal{P}$
- pc_p is the production count of panel $p \in \mathcal{P}$
- t_j is the processing time of job j
- t_{ex} is the time needed for a toolbit exchange
- t_{setup} necessary time to change a trolley
- $vis(j, pm, s) \in \{0, 1\}$ equals "1" iff job $j \in J$ is visible on pick-and-placement module $pm \in \mathcal{M}$ in index step $s \in S$.
- $ct(j)$ denotes the component type $ct \in \mathcal{CT}$ of mounting task $j \in J$
- w_{ct} denotes the width of component type $ct \in \mathcal{CT}$
- w_{pm} denotes the width of pick-and-placement module $pm \in \mathcal{M}$
- $f_{o,ct}^{ct} \in \{0, 1\}$ equals "1" iff toolbit $o \in O$ is compatible with component type $ct \in \mathcal{CT}$

Variables

We use the following variables:

- x_p^{wl} denotes the total workload of panel $p \in \mathcal{P}$
- $x_{p,s}^{wl}$ denotes the workload of panel $p \in \mathcal{P}$ in index step $s \in S$
- $x_{s,pm,p}^{te}$ denotes the number of toolbit exchange operations which are executed in index step $s \in S$ for panel $p \in \mathcal{P}$ on the pick-and-placement module $pm \in \mathcal{M}$
- $x_{j,pm,s}^{mt} \in \{0,1\}$ equals "1" iff mounting task j is executed by the pick-and-placement module $pm \in \mathcal{M}$ in index step $s \in S$
- $x_{p,g} \in \{0,1\}$ equals "1" iff panel $p \in \mathcal{P}$ belongs to group $g \in \mathcal{G}$.
- $x_g \in \{0,1\}$ equals "1" iff group $g \in \mathcal{G}$ is used.
- $x_{p,g,ct,pm} \in \{0,1\}$ equals "1" iff the machine setup has equipped the pick-and-placement module $pm \in \mathcal{DM}$ with component type $ct \in \mathcal{CT}$ for panel $p \in \mathcal{P}$ and p belongs to group $g \in \mathcal{G}$
- $x_{ct,pm} \in \{0,1\}$ equals "1" iff the static pick-and-placement module $pm \in \mathcal{SM}$ is equipped with component type $ct \in \mathcal{CT}$
- $x_{g,ct,pm} \in \{0,1\}$ equals "1" iff the dynamic pick-and-placement module $pm \in \mathcal{DM}$ is equipped with component type $ct \in \mathcal{CT}$ in group $g \in \mathcal{G}$
- $x_{o,s,pm,p}^{t} \in \{0,1\}$ equals "1" iff toolbit $o \in O$ is used in index step $s \in S$ at the pick-and-placement module $pm \in \mathcal{M}$ for panel $p \in \mathcal{P}$
- $x_{o,s,pm,p}^{f} \in \{0,1\}$ equals "1" iff toolbit $o \in O$ is used as the *first* toolbit in index step $s \in S$ at the pick-and-placement module $pm \in \mathcal{M}$ for panel $p \in \mathcal{P}$
- $x_{o,s,pm,p}^{l} \in \{0,1\}$ equals "1" iff toolbit $o \in O$ is used as the *last* toolbit in index step $s \in S$ at the pick-and-placement module $pm \in \mathcal{M}$ for panel $p \in \mathcal{P}$
- $x_{s,pm,p}^{onetbt} \in \{0,1\}$ equals "1" iff exactly one toolbit type is assigned with the combination of index step $s \in S$, pick-and-placement module $pm \in \mathcal{M}$ and panel $p \in \mathcal{P}$
- $x_{o,s,pm,p}^{prec} \in \{0,1\}$ equals "1" iff $o \in O$ is the first *or* the last toolbit associated with pick-and-placement module $pm \in \mathcal{M}$, index step $s \in S$ and panel $p \in \mathcal{P}$
- $x_{o,s,pm,p}^{f \vee l} \in \{0,1\}$ equals "1" iff if toolbit $o \in O$ is the first or the last associated toolbit with pick-and-placement module $pm \in \mathcal{M}$, index step $s \in S$ and panel $p \in \mathcal{P}$
- $x_{s,pm,p}^{f==l} \in \{0,1\}$ equals "1" iff at least two different toolbits are associated with index step $s \in S$, pick-and-placement module $pm \in \mathcal{M}$ and panel $p \in \mathcal{P}$ but the first and last toolbit have equal types.

Bibliography

AGNETIS, A., FILIPPI, C.: (2005). An asymptotically exact algorithm for the high-multiplicity bin packing problem. *Math. Program.*, 104(1), S. 21–37. [12, 13, 17, 39, 70, 76]

AMMONS, J. C., CARLYLE, M., CRANMER, L., DEPUY, G., ELLIS, K. P., MCGINNIS, L. F., TOVEY, C. A., XU, H.: (1997). Component allocation to balance workload in printed circuit card assembly systems. *IIE Transactions*, 29(4), S. 265–275. [121, 122, 136, 139]

APPLEGATE, D., BURIOL, L. S., DILLARD, B. L., JOHNSON, D. S., SHOR, P. W.: (2003). *The Cutting-Stock Approach to Bin Packing: Theory and Experiments*. In *Fifth Workshop on Algorithm Engineering and Experiments (ALENEX)*, S. 1–15, Baltimore, MD, USA. SIAM. [42]

APPLEGATE, D. L., BIXBY, R. E., CHVÁTAL, V., COOK, W. J.: (2007). *The Traveling Salesman Problem: A Computational Study*. Princeton Series in Applied Mathematics. Princeton University Press. [78]

APT, K.: (2003). *Principles of Constraint Programming*. Cambridge University Press. [78]

ATAMTÜRK, A.: (2004). *Polyhedral Methods in Discrete Optimization*. In Hosten, S., Lee, J., Thomas, R. (Hrsg), *Trends in Optimization*, Band 61, S. 21–37. American Mathematical Society. Proceedings of Symposia in Applied Mathematics. [39]

BALAKRISHNAN, A., VANDERBECK, F.: (1999). A Tactical Planning Model for Mixed-Model Electronics Assembly Operations. *Oper. Res.*, 47(3), S. 395–409. [121, 131]

BARVINOK, A., POMMERSHEIM, J.: (1999). An algorithmic theory of lattice points in polyhedra. [45, 49]

BELDICEANU, N.: (2000). *Global Constraints as Graph Properties on a Structured Network of Elementary Constraints of the Same Type*. In *Principles and Practice of Constraint Programming*, S. 52–66. [79]

BELDICEANU, N., CARLSSON, M., DEMASSEY, S., PETIT, T.: (2007). Global Constraint Catalog: Past, Present and Future. *Constraints*, 12(1), S. 21–62. [78]

BELDICEANU, N., CARLSSON, M., PETIT, T.: (2004). Deriving filtering algorithms from constraint checkers. [79]

BELDICEANU, N., CONTJEAN, E.: (1994). Introducing Global Constraints in CHIP. *Mathematical and Computer Modelling*, 12, S. 97–123. [78]

BENTLEY, J.: (1985). Programmimg pearls. *Commun. ACM*, 28(9), S. 896–901. [119]

BIENSTOCK, D.: (1996). Computational Study of a Family of Mixed-Integer Quadratic Programming Problems. *Mathematical Programming*, 74, S. 121–140. [95]

BRUCKER, P.: (2007). *Scheduling Algorithms*. Springer, Berlin, 5 Auflage. [83, 87]

BRUNS, W., GUBELADZE, J., HENK, M., MARTIN, A., WEISMANTEL, R.: (1999). A Counterexample to an Integer Analogue of Carathéodory's Theorem. *Journal für die Reine und Angewandte Mathematik*, 510, S. 179–185. [60]

CHANG, T.-J., MEADE, N., BEASLEY, J., SHARAIHA, Y.: (2000). Heuristics for cardinality constrained portfolio optimisation. *Computers and Operations Research*, 27, S. 1271–1302. [95]

COFFMAN, JR., E. G., GAREY, M. R., JOHNSON, D. S.: (1987). Bin packing with divisible item sizes. *J. Complexity*, 3(4), S. 406–428. [12, 17]

COOK, W., FONLUPT, J., SCHRIJVER, A.: (1986). An integer analogue of Carathéodory's Theorem. *Journal of Combinatorial theory*, 40(B), S. 63–70. [60]

COOK, W., HARTMANN, M., KANNAN, R., MCDIARMID, C.: (1992). On integer points in polyhedra. *Combinatorica*, 12(1), S. 27–37. [49]

CRAMA, Y., VAN DE KLUNDERT, J., SPIEKSMA, F.: (2002). Production planning problems in printed circuit board assembly. *Discrete Appl. Math.*, 123(1-3), S. 339–361. [121]

DANTZIG, G.: (2002). Linear Programming. *Operations Research*, 50(1), S. 42–47. [77]

EISEMANN, K.: (1957). The Trim Problem. *Management Science*, 3(3), S. 279–284. [42]

EISENBRAND, F.: (2003). *Fast Integer Programming in Fixed Dimension*. In *ESA*, Band 2832 von *Lecture Notes in Computer Science*, S. 196–207. Springer. [70, 71]

ELLIS, K., MCGINNIS, L., AMMONS, J.: (2003). An approach for grouping circuit cards into families to minimize assembly time on a placement machine. *IEEE Transactions on Electronics Packaging Manufacturing*, 26(1), S. 22–30. [121]

FAHLE, T., SELLMANN, M.: (2002). Cost-Based Filtering for the Constrained Knapsack Problem. *Annals of Operations Research*, 115, S. 73–92. [84]

FILGUEIRAS, M., TOMÁS, A. P.: (1993). *Fast Methods for Solving Linear Diophantine Equations*. In *EPIA '93: Proceedings of the 6th Portuguese Conference on Artificial Intelligence*, S. 297–306, London, UK. Springer-Verlag. [72]

FILIPPI, C.: (2007). On the Bin Packing Problem with a Fixed Number of Object Weights. *European Journal of Operational Research*, 181(1), S. 117–126. [39]

FOCACCI, F., LODI, A., MILANO, M.: (2002). Optimization-Oriented Global Constraints. *Constraints*, 7, S. 351–365. [78, 81]

FRANK, A., TARDOS, E.: (1987). An application of simultaneous Diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1), S. 49–65. [71]

GAREY, M., JOHNSON, D.: (1979). *Computers and Intractability: A guide to the Theory of $\mathcal{NP}$–completeness*. W.H. Freeman and Co. [12, 40, 81, 83, 85, 126, 131, 132]

GAREY, M., JOHNSON, D.: (1985). A 71/60 theorem for bin packing. *Journal of Complexity*, 1, S. 65–106. [12]

GAU, T., WÄSCHER, G.: (1996). Heuristics for the integer one-dimensional cutting stock problem: A computational study. *OR Spektrum*, 18(3), S. 131–144. [39, 42]

GILES, R., PULLEYBLANK, W.: (1979). Total dual integrality and integer polyhedra. *Linear algebra and its applications*, 25, S. 191–196. [60]

GILMORE, P. C., GOMORY, R. E.: (1961). A linear programming approach to the cutting stock problem. *Operations Research*, 9, S. 849–859. [19, 39, 42]

GILMORE, P. C., GOMORY, R. E.: (1963). A linear programming approach to the cutting stock problem, part II. *Operations Research*, 11, S. 863–888. [39, 42]

Goodman, J. E., O'Rourke, J. (Hrsg): (1997). *Handbook of discrete and computational geometry*. CRC Press Inc., Boca Raton, FL, USA. [45]

GORDAN, P.: (1873). Über die Auflösung linearer Gleichungen mit reellen Coefficienten. *Math. Ann.*, 8, S. 23–28. [60]

GRANDONI, F., ITALIANO, G. F.: (2006). *Algorithms and Constraint Programming*. In *Principles and Practice of Constraint Programming - CP 2006*, S. 2–14, Berlin. Springer. [79]

GRÖTSCHEL, M., LOVÁSZ, L., SCHRIJVER, A.: (1993). *Geometric Algorithms and Combinatorial Optimization*, Band 2 von *Algorithms and Combinatorics*. Springer, second corrected edition Auflage. [70, 71]

HAASE, C.: (2000). *Lattice Polytopes and Triangulations*. Doktorarbeit, Mathematics, TU Berlin. [45]

HIRSCHBERG, D. S., WONG, C. K.: (1976). A Polynomial-Time Algorithm for the Knapsack Problem with Two Variables. *J. ACM*, 23(1), S. 147–154. [17]

HOCHBAUM, D.: (1997). *Approximation Algorithms for $\mathcal{NP}$-hard Problems*. PWS Publishing Company, Boston, MA. [17, 19]

HOCHBAUM, D., SHAMIR, R.: (1991). Strongly polynomial algorithms for the high multiplicity scheduling problem. *Operations Research*, 39, S. 648–653. [19]

HOOKER, J.: (2000). *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. Wiley. [78, 81]

HOROWITZ, E., SAHNI, S.: (1978). *Fundamentals of Computer Algorithms*. Computer Science Press, Rockville, MD, USA. [136]

HUTTER, M., MASTROLILLI, M.: (2006). Hybrid rounding techniques for knapsack problems. *Discrete Appl. Math.*, 154(4), S. 640–649. [17, 19]

JOHNSSON, M., SMED, J.: (2001). *Observations on PCB Assembly Optimization*. In *The APEX Files: Proceedings of Technical Conference*, S. SM6–3 1–6, San Diego, CA. IPC SMEMA Council. [121, 123]

KANTOROVICH, L.: (1960). Mathematical methods of organising and planning production. *Management Science*, 6, S. 366–422. [41]

KARMARKAR, N., KARP, R. M.: (1982). *An Efficient Approximation Scheme for the One-Dimensional Bin-Packing Problem*. In *23rd Annual Symposium on Foundations of Computer Science, 3-5 November 1982, Chicago, Illinois, USA*, S. 312–320. IEEE. [39]

KARP, R. M.: (1972). *Reducibility among combinatorial problems*. Plenum Press, New York. [17, 85, 131]

KELLERER, H., PFERSCHY, U., PISINGER, D.: (2004). *Knapsack Problems*. Springer, Heidelberg. [83, 84]

KNUUTILA, T., HIRVIKORPI, M., JOHNSSON, M., NEVALAINEN, O.: (2004). Grouping PCB Assembly Jobs with Feeders of Several Types. *Journal of Flexible Manufacturing Systems*, 16(2). [120]

KNUUTILA, T., HIRVIKORPI, M., JOHNSSON, M., NEVALAINEN, O.: (2005). Grouping of PCB Assembly Jobs in the Case of Flexible Feeder Units. *Journal of Engineering Optimization*, 37(1), S. 29–48. [120]

LEMAIRE, P., FINKE, G., BRAUNER, N.: (2003a). *Multibin Packing with the Best-Fit Rule*. In *Proceedings of the 1st Multidisciplinary International Conference on Scheduling : Theory and Applications*, Band 1, S. 74–88, Nottingham, England. [124, 125]

LEMAIRE, P., FINKE, G., BRAUNER, N.: (2003b). *Packing of Multibin Objects*. In *Proceedings of the International Conference on Industrial Engineering and Production Management*, Band 1, S. 422–431, Porto, Portugal. [124]

LEMAIRE, P., FINKE, G., BRAUNER, N.: (2005). *The Best-Fit Rule For Multibin Packing: An Extension of Graham's List Algorithms*, S. 269–286. Springer US. [136]

LEMAIRE, P., FINKE, G., BRAUNER, N.: (2006). Models and Complexity of Multibin Packing Problems. *Journal of Mathematical Modelling and Algorithms*, 5(3), S. 353–370. [124]

LEON, V., PETERS, B.: (1996). Replanning and analysis of partial setup strategies in printed circuit board assembly systems. *International Journal of Flexible Manufacturing Systems*, 8(4), S. 289–411. [121]

LOFGREN, C., MCGINNIS, L.: (1986). *Dynamic Scheduling for Flexible Printed Circuit Card Assembly*. In *Proceedings IEEE International Conference on Systems, Man, and Cybernetics*, Band 3, S. 1294–1297, Atlanta, Georgia. [121]

LÜSCHER, M.: (1994). A portable high-quality random number generator for lattice field theory calculations. *Computer Physics Communications*, 79, S. 100Ű110. [31]

MAGAZINE, M., POLAK, G., SHARMA, D.: (2002). A Multi-exchange Neighborhood Search Heuristic for PCB Manufacturing. *Journal of Electronis Manufacturing*, 11(2), S. 107–119. [120]

MAHER, M.: (2002). *Propagation Completeness of Reactive Constraints*. In *18th International Conference on Logic Programming*, S. 148–162. [79]

MARCOTTE, O.: (1986). An instance of the cutting stock problem for which the rounding property does not hold. *Operations Research Letters*, 4, S. 239–243. [40]

MARKOWITZ, H.: (1952). Portfolio selection. *Journal of Finance*, 7, S. 77–91. [95]

MARSAGLIA, G., TSANG, W.: (2000). The Ziggurat Method for Generating Random Variables. *Journal of Statistical Software*, 5(8), S. 1–7. [31]

MATSUMOTO, M., NISHIMURA, T.: (1998). Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1), S. 3–30. [31]

MCCORMICK, S. T., SMALLWOOD, S. R., SPIEKSMA, F. C. R.: (2001). A Polynomial Algorithm for Multiprocessor Scheduling with Two Job Lengths. *Math. Oper. Res.*, 26(1), S. 31–49. [12, 13, 17, 39, 44, 76]

MEHLHORN, K., THIEL, S.: (2000). *Faster Algorithms for Bound-Consistency of the Sortedness and the Alldifferent Constraint*. In *Principles and Practice of Constraint Programming*, S. 306–319. [79]

MILANO, M.: (2003). *Constraint and Integer Programming: Toward a Unified Methodology (Operations Research/Computer Science Interfaces, 27)*. Kluwer Academic Publishers, Norwell, MA, USA. [77]

MÜLLER-HANNEMANN, M., TAZARI, S., WEIHE, K.: (2006). *Workload Balancing in Multi-stage Production Processes*. In *Proc. of the 5th Int. Workshop on Experimental Algorithms*, Band 4007, S. 49–60. Springer LNCS. [121, 139]

NEMHAUSER, G., WOLSEY, L.: (1988). *Integer and Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley. [39, 40, 83]

REGIN, J.-C., RUEHER, M.: (2000). *A Global Constraint Combining a Sum Constraint and Difference Constraints*. In *Principles and Practice of Constraint Programming*, S. 384–395. [78]

REINELT, G.: (1991). TSPLIB – A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3(4), S. 376–384. [30, 31, 36, 37]

REINELT, G.: (1994). *The Traveling Salesman – Computational Solutions for TSP Applications*, Band 840 von *Lecture Notes in Computer Science*. Springer. [78]

SALONEN, K., JOHNSSON, M., SMED, J., JOHTELA, T., NEVALAINEN, O.: (2000). *A Comparison of Group and Minimum Setup Strategies in PCB Assembly*. In Hernández, Süer (Hrsg), *Proceedings of Group Technology/Cellular Manufacturing World Symposium—Year 2000*, S. 95–100, San Juan, Puerto Rico. [120]

SALONEN, K., JOHNSSON, M., SMED, J., NEVALAINEN, O.: (2003). *Job Grouping with Minimum Setup in PCB Assembly*. In Sormaz, Süer (Hrsg), *Proceedings of the Group Technology/Cellular Manufacturing World Symposium–Year 2003*, S. 221–225, Columbus, OH. [120]

SCHEITHAUER, G., TERNO, J.: (1995). A branch&bound algorithm for solving one–dimensional cutting stock problems exactly. *Applicationes Mathematicae*, 23(2), S. 151–167. [39]

SCHRIJVER, A.: (1987). *Theory of Linear and Integer Programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley. [39]

SCHRIJVER, A.: (2003). *Combinatorial Optimization – Polyhedra and Efficiency*. Algorithms and Combinatorics. Springer. [39]

SCHRIJVER, A., SEYMOUR, P., WINKLER, P.: (1998). The Ring Loading Problem. *SIAM Journal on Discrete Mathematics*, 11(1), S. 1–14. [124]

SEBÖ, A.: (1990). *Hilbert Bases, Caratheodory's Theorem and Combinatorial Optimization*. In *Proceedings of the 1st Integer Programming and Combinatorial Optimization Conference*, S. 431–455, Waterloo, Ont., Canada, Canada. University of Waterloo Press. [60]

SELLMANN, M.: (2003). *Approximated Consistency for Knapsack Constraints*. In *9th International Conference of Principles and Practice of Constraint Programming (CP 2003)*, Band 2833 von *Lecture Notes in Computer Science*, S. 679–693. Springer. [79, 84]

SIMONIS, H.: (2007). Models for Global Constraint Applications. *Constraints*, 12(1), S. 63–92. [79, 83]

SMED, J., SALONEN, K., JOHNSSON, M., JOHTELA, T., NEVALAINEN, O.: (2000). *A Comparison of Group and Minimum Setup Strategies in PCB Assembly*. Technischer Bericht 327, Turku Centre for Computer Science. [120]

STEINBACH, M., KARYPIS, G., KUMAR, V.: (2000). *A comparison of document clustering techniques*. Technischer Bericht, KDD Workshop on Text Mining. [120, 135]

TRICK, M.: (2001). *A Dynamic Programming Approach for Consistency and Propagation for Knapsack Constraints*. In *Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, S. 113–124. [84]

VAN DER CORPUT, J.: (1931). Über Systeme von linear-homogenen Gleichungen und Ungleichungen. *Proceedings Koninklijke Akademie van Wetenschappen te Amsterdam*, 34, S. 368–371. [60]

VANCE, P. H., BARNHART, C., JOHNSON, E. L., NEMHAUSER, G. L.: (1994). Solving binary cutting stock problems by column generation and branch-and-bound. *Comput. Optim. Appl.*, 3(2), S. 111–130. [42]

VAZIRANI, V.: (2001). *Approximation Algorithms*. Springer. [12]

VERHAEGH, W. F. J., AARTS, E. H. L.: (1997). A polynomial-time algorithm for knapsack with divisible item sizes. *Inf. Process. Lett.*, 62(4), S. 217–221. [17]

WOLSEY, L.: (1998). *Integer Programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley. [39]

List of Figures

[1] Image by courtesy of Philips Assembléon.
[2] Image by courtesy of Philips Assembléon.

List of Tables

List of Algorithms

Index

Symbol Index

$\mathbb{N}$	set of natural numbers $(1, 2, 3, \ldots)$
$\mathbb{N}$	set of natural numbers including zero $(0, 1, 2, 3, \ldots)$
$\mathbb{Q}$	set of rational numbers $(\frac{3}{2}, -\frac{5}{7}, 2)$
$\mathbb{R}$	set of real numbers $(1.2345, \frac{1}{3}, \pi, \sqrt{2}, \ldots)$
$\mathbb{Z}$	set of integral numbers $(\ldots, -2, -1, 0, 1, 2, 3, \ldots)$
$\mathbb{Z}^\star$	set of nonnegative integral numbers $(0, 1, 2, 3, \ldots)$
$\mathbb{R}^m$	real vector space $\mathbb{R} \times \ldots \times \mathbb{R}$ (m–times) of dimension m
$\mathbb{Z}^m$	canonical integral lattice $\mathbb{Z} \times \ldots \times \mathbb{Z}$ (m–times) of dimension m
$\mathbb{Z}_+^m$	canonical nonnegative integral lattice $\mathbb{Z}^\star \times \ldots \times \mathbb{Z}^\star$ (m–times) of dimension m
$\oplus, \ominus$	componentwise operations on heteroary codings $\mathcal{H}^d$
∇	gradient
$\lfloor \cdot \rfloor$	next smaller integral number
$\lceil \cdot \rceil$	next greater integral number
$\lvert \cdot \rvert$	Euclidean norm
$\pi(i)$	assignment function $\pi : \mathbb{I} \longrightarrow \mathbb{B}$ from set of items $\mathbb{I}$ to set of bins $\mathbb{B}$
$\boldsymbol{v}$	vector in $\mathbb{K}^m$
$\dim(\boldsymbol{v})$	dimension m of $\boldsymbol{v}$
$\boldsymbol{A}$	$m \times n$ matrix $\boldsymbol{A} := (a_{ij}) \in, i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}$
$\det(\boldsymbol{A})$	determinant of matrix $\boldsymbol{A}$
$\operatorname{rank}(\boldsymbol{A})$	rank of matrix $\boldsymbol{A}$
$\boldsymbol{I}$	identity matrix with $a_{ij} = 1$ for $i \equiv j$, and 0 otherwise
$\boldsymbol{0}$	zero vector or matrix
$\mathcal{S}$	vertex set $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_n\} \subset \mathbb{K}^m$
$\operatorname{aff}(\mathcal{S})$	affine hull over all $\boldsymbol{f}_\ell \in \mathcal{S}$
$\lvert \mathcal{S} \rvert$	cardinality of $\mathcal{S}$
$\operatorname{cone}(\mathcal{S})$	conical hull over all $\boldsymbol{f}_\ell \in \mathcal{S}$
$\operatorname{conv}(\mathcal{S})$	convex hull over all $\boldsymbol{f}_\ell \in \mathcal{S}$
$\operatorname{lin}(\mathcal{S})$	linear hull over all $\boldsymbol{f}_\ell \in \mathcal{S}$
$\operatorname{diam}(\mathcal{S})$	diameter of a vertex set $\mathcal{S}$

$\mathrm{dom}(\mathcal{S})$	convex hull over the vertex set $\mathcal{S}$ and all vertices dominated by $\mathcal{S}$
$\mathrm{pos}(\mathcal{S})$	rational polyhedral cone given by the positive combinations of $\boldsymbol{f}_\ell \in \mathbb{Z}^m$ from $\mathcal{S}$
$\underset{A^w}{\Delta} a_i$	weighted absolute rounding error at index i
$\underset{R^w}{\Delta} a_i$	weighted relative rounding error at index i
Δ_k	offset in interval $[I_{k-1}, I_k]$ (Adaptive Rounding)
$\kappa_\mu(\lambda)$	increment in update formula (Adaptive Rounding)
$\kappa_\mu(\lambda_1, \lambda_2)$	increment in equilibrium situation (Adaptive Rounding)
$\kappa_{\mu\mu}$	minimum increment in update formula (Adaptive Rounding)
$\tilde{a}_i$	rounding value at index i
$\mathcal{B}$	a general HIGH MULTIPLICITY BIN PACKING (HMBP) instance
$\tilde{\mathcal{B}}$	(HMBP) instance that is equivalent to $\mathcal{B}$
$\boldsymbol{C}$	vector of bin sizes in non-uniform case, C bin size in uniform case
$\mathcal{C}(\mathcal{S})$	(pointed) cone denoted by all non-negative (real) combinations of vectors in $\mathcal{S}$
$\mathcal{C}(\mathcal{F})$	(pointed) convex lattice cone defined by $\mathrm{conv}(\mathcal{F}, 0)$
$C_{\max}$	maximum of absolute/relative weighted rounding errors
C_Σ	sum of absolute/relative weighted rounding errors
$\mathcal{D}$	fundamental domain/parallelepiped for the lattice Λ
$\mathcal{E}$	absolute/relative error function
$\mathcal{F}_\ell$	characteristic facet of the (pointed) convex lattice cone $\mathcal{C}_\ell$
$\mathcal{G}$	set of spanning dominating bin patterns of $\mathcal{P}^\star$
G	directed graph $D = (V, A)$ with set of nodes V and set of arcs A
G	undirected graph $G = (V, E)$ with set of nodes V and set of edges E
$\mathcal{H}(\mathcal{C})$	finite generating system of $\mathcal{C} \cap \mathbb{Z}^m$ (Hilbert basis)
$\mathcal{H}^d$	heteroary coding of dimension d
K	maximum number of rounding values/intervals
K_1	number of rounding intervals for Adaptive Rounding
K_2	number of rounding values per interval
Λ	lattice in $\mathbb{R}^m$
$\tilde{L}$	set of dominating bin patterns for an instance $\mathcal{B}$
L	set of all bin patterns for an instance $\mathcal{B}$
L_j	lower bound on the load of bin j
$\mathcal{M}$	matroid $\mathcal{M}$ with rank $rank(\mathcal{M})$
$\mathcal{P}$	convex polytope given by $\{\boldsymbol{x} \in \mathbb{R}^m : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b};\, \boldsymbol{x} \geq 0\}$

$\partial\mathcal{P}$	border of a polytope or convex set $\mathcal{P}$
$\mathcal{P}^\star$	lattice polytope in Λ
$\boldsymbol{s}$	bin pattern in the canonical integral lattice $\mathbb{Z}^m$
S_j	simplex defined by the axes and the hyperplane $\boldsymbol{w}^\intercal \boldsymbol{x} \equiv C_j$
U_j	upper bound on the load of bin j
$\boldsymbol{v}$	vector of item multiplicities
$\boldsymbol{w}$	vector of item sizes
$\boldsymbol{w}_i$	generators of the lattice Λ
X_k	first rounding value in interval $[I_{k-1}, I_k]$ (Adaptive Rounding)